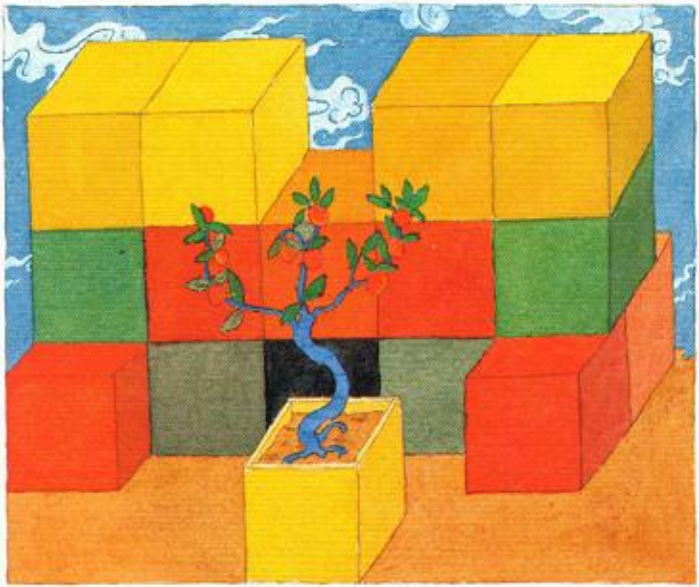
TO 7 MO5 TO7

FORTH, MANUEL DE REFERENCE

SEFI



TO TEK International

cedic
nathan

Manuel de référence du FORTH

S.E.F.I.

TO TEK International
cedic nathan

Couverture : Olivier Leclercq

Illustrations : Olivier Leclercq et Roland Carlier

Maquette : A. Dufourcq et P. Deiller

Ce volume porte la référence
ISBN 2-7124-0576-5

Toute reproduction, même partielle, de cet ouvrage est interdite. Une copie ou reproduction par quelque procédé que ce soit, photographie, photocopie, microfilm, bande magnétique, disque ou autre, constitue, une contrefaçon passible des peines prévues par la loi du 11 mars 1957 sur la protection des droits d'auteur.

© CEDIC 1985
CEDIC, 6-10 boulevard Jourdan, 75014 - Paris

Sommaire

Introduction 5

Première partie
Glossaire général des mots de la cartouche ROM

1. Manipulations sur les piles 9
2. Calculs arithmétiques et logiques 19
3. Structures de contrôle 35
4. Définition de nouveaux mots
   Compilation et interprétation 49
5. Chaînes de caractères
   Conversions entre données numériques et alphanumériques 73
6. Graphique et couleur 93
7. Musique 109
8. Opérations sur la mémoire 119
9. Entrées-sorties (sauf cassettes et disquettes) 137
10. Stockage de masse : cassettes et disquettes 159
11. Utilitaires et mot divers 179

Deuxième partie
Compléments et annexes

A. L'éditeur 193
B. Les mots de la disquette-système FORTH 203
C. Structure du dictionnaire 217
D. Les mots système. Utilisation de la vectorisation 229
E. Organisation de la mémoire 233
F. Mise en service 241
G. Glossaire des mots et abréviations utilisés 243
H. Index des mots du FORTH 253
I. Codes ASCII et messages d'erreur 259

Introduction

Comment utiliser ce manuel

Ce manuel est un manuel de référence, et n'a pas pour objet de vous apprendre à programmer en FORTH. Si vous abordez l'étude de ce langage, vous pouvez vous reporter d'abord au manuel d'initiation, ou à un autre ouvrage de présentation de FORTH.
Dans ce cas, ou si vous connaissez déjà FORTH, consultez l'annexe F pour le démarrage du système, la description des touches, et les particularités de votre appareil.

Les onze premiers chapitres décrivent les mots FORTH de la cartouche ROM, regroupés par thèmes. Ces regroupements ne sont pas toujours faciles (certains mots sont à cheval entre plusieurs chapitres...), et sont parfois arbitraires. Si vous ne trouvez pas un mot là où vous l'espériez, consultez l'index (annexe H, en fin de manuel). Il vous dira quel chapitre consulter.
Nous avons choisi de faire figurer en annexe l'utilisation de l'éditeur (qui n'est pas à proprement parler du FORTH), ainsi que les mots de la disquette-système, qui ne concernent que les possesseurs de lecteurs de disquettes.
Si une explication fait intervenir un terme qui vous est étranger, le lexique de l'annexe G pourra vous aider à en comprendre la signification. Il ne s'agit pas d'une encyclopédie, mais il pourra dans la plupart des cas orienter votre recherche...

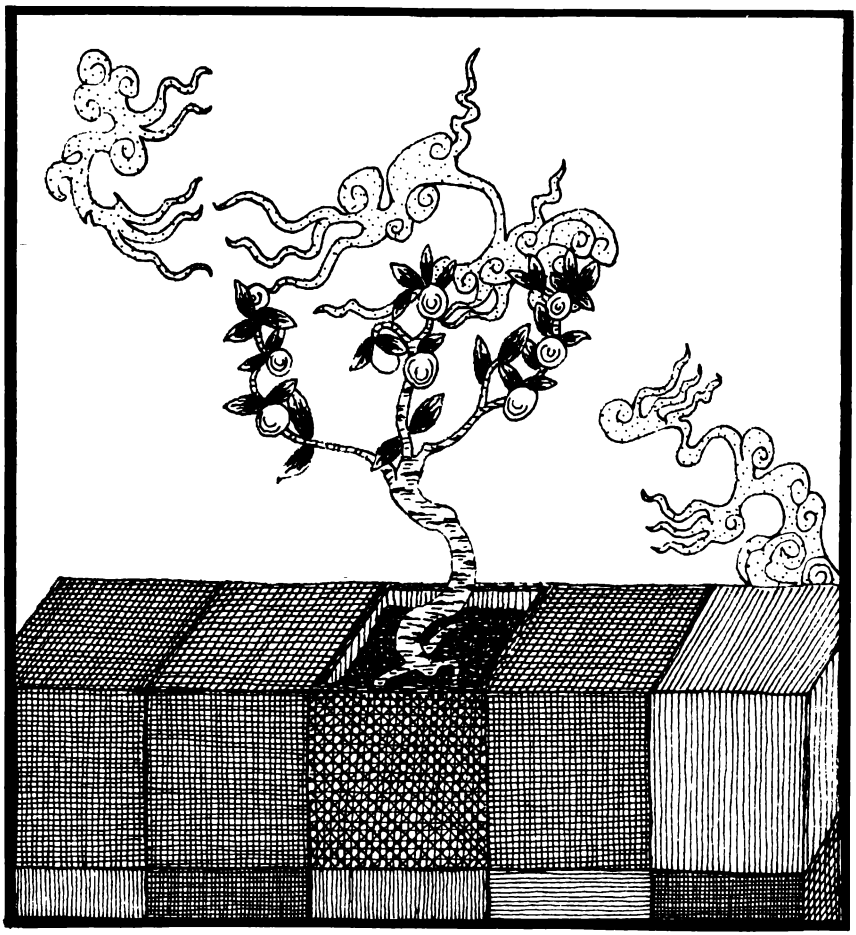
Enfin, si vous êtes curieux et si vous vous sentez à l'aise en FORTH, les annexes C, D, et E vous aideront à tirer le meilleur parti possible du langage et du matériel. Vous y trouverez des renseignements sur la structure et l'organisation du langage, et sur les possibilités de personnalisation de votre système.

Ce manuel comporte de nombreux exemples, et beaucoup de renvois. N'hésitez pas dans un premier temps à taper les exemples et à suivre les renvois. Vous apprendrez vite quels sont les mots fondamentaux du langage, et comment les mots d'une même famille interagissent entre eux.
FORTH comporte un grand nombre d'instructions (environ cinq cents, contre une centaine pour les autres langages courants), et il n'est pas facile (ni souhaitable) de les connaître tous. La meilleure approche est d'apprendre, pour commencer, quelques mots fondamentaux pour écrire des applications simples, et d'agrandir progressivement votre "vocabulaire" au fur et à mesure de vos investigations dans le manuel de référence.

Dans cet ordre d'idées, ce manuel n'est pas conçu pour favoriser une lecture séquentielle...
Les exemples sont indépendants d'un mot à l'autre (sauf renvoi). En particulier, les exemples qui font intervenir une visualisation de l'état de la pile (par le mot .S) supposent que la pile était vide...
Il y a plusieurs possibilités pour réinitialiser les piles avant la frappe d'un exemple : le plus simple est d'utiliser le mot ABORT, ou de provoquer une erreur en tapant un mot inconnu... Par exemple MMM (suivi de la touche ENTREE). Vous obtiendrez un message d'erreur "UK ERROR", mais vos piles seront réinitialisées...

Première partie

Glossaire général des mots de la cartouche ROM



Chapitre 1

Manipulations sur les piles

FORTH emploie deux piles :
— La pile "système", ou pile des retours, qui lui sert le plus souvent à mémoriser l'adresse du prochain mot à traiter ("au retour du traitement présent"; pour plus de détails sur le fonctionnement du système, voir l'annexe C), mais qu'il utilise aussi dans des mots du genre DO...LOOP (voir chapitre 3). Dans certains cas, vous pouvez avoir intérêt à utiliser aussi cette pile pour ranger temporairement des nombres, à vos risques et périls (Voir >R, R>, R@).
— La pile "utilisateur", ou pile des données (en anglais "parameter stack"), qu'il laisse à votre disposition. C'est sur cette pile que la plupart des mots FORTH vont chercher leurs paramètres, ou déposent le résultat de leur action (dans le cas de nombres ou d'adresses). Lorsqu'on parle de "pile" sans précision, il s'agit de la pile des données.

La pile est composée de "cellules" de 16 bits (deux octets). Chacune de ces cellules peut contenir un nombre entier, qui peut être interprété comme nombre signé ou non signé, ou comme une adresse. Ces cellules peuvent être groupées par deux (quatre octets), et former des nombres "double longueur", signés ou non.

Les conventions utilisées pour représenter l'effet apparent des mots sur la pile sont les suivantes :

— en une cellule (deux octets) :

n	...	nombre entier signé (entre −32768 et 32767)
un	...	nombre entier non signé (entre Ø et 65535)
adr	...	nombre entier représentant une adresse
cond	...	nombre exprimant une valeur du type "vrai" ou "faux". Ø signifie "faux", toute autre valeur signifie "vrai".
c	...	caractère. En fait, un nombre de Ø à 255 représentant le code ASCII étendu d'un caractère.

— en deux cellules (quatre octets) :

d	..	nombre double longueur signé (entre −2147483648 et 2147483647)
ud	...	nombre double longueur non signé (entre Ø et 4294967295)
str	...	chaîne de caractères (en fait deux nombres entiers, adr et l) adr est l'adresse du premier caractère de la chaîne. l est la longueur (nombre de caractères de la chaîne).

Ces mots sont décrits par l'effet apparent qu'ils ont sur la pile. En réalité, la pile est "tête en bas" en mémoire, c'est-à-dire que son sommet est à l'adresse la plus basse. D'autre part, elle n'est jamais effacée, c'est simplement un pointeur de "sommet de pile" qui change de valeur (lorsqu'on exécute par exemple le mot DROP, la valeur du pointeur de pile est augmentée de 2, mais la valeur qui était au "sommet" de la pile n'est pas enlevée. Elle sera remplacée par la prochaine valeur qui viendra sur la pile...).

Ø 1 2 3 --- n

Constantes laissant sur la pile la valeur Ø, 1, 2, ou 3.
Ces nombres étant souvent utilisés, il est intéressant de les définir dans le dictionnaire comme constantes. On y gagne à la fois en place mémoire et en rapidité d'exécution.

```
2 BASE ! 3 . 11 OK
```

. . n1 --- n2

Echange les octets de poids fort et de poids faible de n1.

```
HEX 1234 > < . 3412 OK
```

>R n ---

Transfère n sur la pile des retours. Cette pile étant utilisée par le système, chaque >R doit absolument être associé à un R> dans le même mot, et même dans la même structure de contrôle (par exemple, la boucle DO utilise aussi la pile des retours pour ranger l'indice de boucle).

Voir aussi R> R@

Exemple : échanger le deuxième et le troisième nombre de la pile...

```
: SWAWAP ( n1 n2 n3 --- n2 n1 n3 ) >R SWAP R> ;
```

est plus rapide et plus compact que "ROT ROT SWAP ROT".

?DUP n --- n (n)

Duplique n s'il n'est pas nul.

Equivalent à "DUP IF DUP THEN", à cela près que cette dernière séquence ne peut être utilisée qu'en mode compilation (car elle contient IF et THEN)...

Surtout utilisé en interprétant n comme une valeur de vérité, dans les cas où on a un traitement à réaliser (qui a besoin du paramètre n) dans les cas où n n'est pas nul.

Exemple : ajouter une liste de nombres commençant (au clavier) par Ø

```
: ++ ( Ø n1 n2 n3 ... np --- somme )
  Ø                    ( Au début, le total vaut Ø )
  BEGIN SWAP
  ?DUP                 ( ?DUP duplique le nombre pour WHILE )
  WHILE +              ( Tant que pas nul, ajouter au total précédent )
  REPEAT ;
Ø 1 3 5 7 ++ . 16 OK  Ø 1 3 5 7 ++  . 25 OK
```

DDROP d ---

Supprime le nombre 32 bits au sommet de la pile. Cette commande sert souvent aussi à supprimer deux nombres 16 bits (ou par exemple les paramètres d'une chaîne de caractères).

```
23 45. DDROP . 23 OK   2 3 4 DDROP . 2 OK
10 STRING XX 20 STRING YY " azerty" XX $! " qwerty" YY $!
XX YY DDROP TYPE azerty OK
```

DDUP d --- d d

Duplique le nombre 32 bits du sommet de la pile. Cette commande sert souvent aussi à dupliquer deux nombres 16 bits (ou par exemple les paramètres d'une chaîne de caractères).

```
1234. DDUP D* D. 1522756 OK
10 STRING XX " azerty "  XX $!
XX DDUP LEN . TYPE 6 azerty OK
```

DEPTH --- n

Donne le nombre de nombres 16 bits contenus dans la pile (avant que n y figure).

```
23 45 678 DEPTH . 3 OK ( en supposant que votre pile était vide...)
```

Puis 23. DEPTH . 5 OK (23. occupe la place de deux nombres 16 bits)

DOVER d1 d2 --- d1 d2 d1

Recopie au sommet de la pile le deuxième nombre 32 bits à partir du sommet précédent. Souvent utilisé pour recopier des paires de nombres 16 bits, ou des paramètres de chaînes de caractères.

```
12. 23. DOVER .S
```

La pile contient 12. 23. 12. .S affiche donc :

```
12 0 23 0 12 0
1 2 3 4 DOVER . . . . . 2 1 4 3 2 1 OK          ( Attention à l'ordre !...)
```

DROP n ---

Enlève (dépile) le nombre 16 bits du sommet de la pile.

```
12 13 DROP . 12 OK
```

DROT d1 d2 d3 --- d2 d3 d1

Extrait le troisième nombre 32 bits à partir du sommet de la pile, et le replace au dessus après avoir décalé les deux nombres 32 bits qui le précédaient. Une autre manière de décrire cette action est de dire que DROT effectue une permutation circulaire sur les trois nombres 32 bits du sommet de la pile, amenant le troisième en première position (l'autre permutation circulaire possible est réalisée par "DROT DROT").

DROT est aussi souvent utilisé pour permuter des nombres 16 bits par "paquets de deux" (il agit alors sur 6 nombres à la fois).

```
1 2 3 4 5 6 DROT .S     La pile affiche : 3 4 5 6 1 2
23. 24. 25. DROT .S     La pile affiche : 24 Ø 25 Ø 23 Ø
1Ø STRING A 1Ø STRING B 1Ø STRING C
   " aaa" A $!  " bbb" B $!  " ccc" C $!
A B C DROT TYPE TYPE TYPE aaacccbbb OK
                                ( au lieu de "cccbbbaaa" sans DROT)
```

DSWAP d1 d2 --- d2 d1

Echange les deux nombres 32 bits du sommet de la pile. Utilisé aussi pour échanger des paires de nombres 16 bits (par exemple des paramètres de chaînes de caractères).

```
1 2 3 4 DSWAP .S    La pile montre : 3 4 1 2
12. 23. DSWAP D. D. 12 23 OK  ( et non pas 23 12)
1Ø STRING A 1Ø STRING B  " azerty" A $! " qwerty" B $!
A B DSWAP TYPE TYPE  azertyqwerty OK
```

DUP n --- n n

Duplique le nombre du sommet de la pile.

```
12 DUP . . 12 12 OK
```

HILO n1 --- n2 n3

Sépare le nombre 16 bits n1 en deux parties n2 et n3, correspondant aux deux octets qui composent n1. n2 est la valeur de l'octet de poids faible, et n3 est la valeur de l'octet de poids fort.

```
HEX 1234 HILO . .  12 34 OK
DECIMAL 10000 HILO .S  16 39 OK  ( 10000 = 39 x 256 + 16)
```

OVER n1 n2 --- n1 n2 n1

Recopie au sommet de la pile le nombre 16 bits qui était en deuxième position à partir du sommet.

```
1 2 OVER .S
```

La pile montre : 1 2 1
DDUP peut très bien être défini par OVER OVER ...

PICK n1 --- n2

Donne la valeur du n1-ième nombre à partir du sommet de la pile, sans compter n1 lui-même. Par exemple, "1 PICK" équivaut à DUP, "2 PICK" équivaut à OVER.

```
12 13 14 15 4 PICK .  12 OK
.S    La pile affiche encore : 12 13 14 15...
```

RØ --- adr

Donne l'adresse initiale de la pile des retours (base de la pile, qui en réalité est l'adresse la plus haute, puisque les piles s'étendent vers les adresses basses).
R est la valeur de la variable "user" RPØ. Le fait de mettre RØ en variable permet de reconfigurer le système.

R > --- n

Transfère (dépile) le nombre du sommet de la pile des retours au sommet de la pile des données. En principe, on n'a à utiliser R > qu'en liaison avec > R.
Exemple : voir > R

Recopie le nombre du sommet de la pile des retours sur la pile des données, sans modifier la pile des retours. Souvent utilisé lorsqu'on se sert de la pile des retours pour ranger temporairement des données (ne pas oublier de dépiler ces données, par exemple par R >, avant que le système n'ait à utiliser la pile des retours).
Exemple : calcul de $a^2 + 3ab + 7b^3$

```
: a2+3ab+7b3          ( a b --- résultat)
    > R               ( Range b sur la pile des retours)
    DUP DUP *         ( Calcul de a²)
    SWAP R@ * 3 *     ( Calcul de 3ab)
    R@ R@ * R> * 7 *
    + + ;             ( Calcul de 7b³ et somme des 3 termes)
```

Noter l'utilisation de R > en dernier, pour restaurer l'état de la pile des retours à la fin de l'exécution.

ROLL n1 n2 n3 ... np p --- n2 n3 ... np n1

Extrait le p-ième nombre à partir du sommet de la pile (p lui-même non compris), et le replace sur le sommet de la pile après avoir décalé les autres nombres de la pile.
Par exemple :

"1 ROLL" revient à ne rien faire.
"2 ROLL" équivaut à SWAP.
"3 ROLL" équivaut à ROT.

```
12 13 14 15 16 4 ROLL .S
PARAMETER STACK: 12 14 15 16 13
```

ROT n1 n2 n3 --- n2 n3 n1

Extrait le troisième nombre 16 bits à partir du sommet de la pile, et le replace au-dessus après avoir décalé les deux nombres 16 bits qui le précédaient. Une autre manière de décrire cette action est de dire que ROT effectue une permutation circulaire sur les trois nombres 16 bits du sommet de la pile, amenant le troisième en première position (l'autre permutation circulaire possible est réalisée par "ROT ROT").
ROT est aussi à utiliser pour faire l'équivalent de SWAP dans le cas

où le nombre du sommet de la pile est un nombre 32 bits (ou un ensemble de paramètre de chaîne de caractères), et que le nombre qui le suit est un nombre 16 bits (l'action de ROT se décrit alors par : n d --- d n)

```
12 13 14 ROT .S     La pile affiche : 13 14 12
23 24. ROT . D. 23 24 OK
```

RP! ---

Réinitialise le pointeur de la pile des retours à la valeur contenue dans la variable user RPØ. Ce mot est utilisé par ABORT, ou pour nettoyer la pile des retours lorsqu'on veut rendre la main au clavier.

RPØ --- **adr**

Variable contenant l'adresse initiale du pointeur de la pile des retours. En modifiant sa valeur (et en réinitialisant le pointeur courant de la pile des retours par RP!), on peut donc reconfigurer le système. D'origine, la pile des retours est limitée à 128 octets, ce qui suffit pour pratiquement toutes les applications. Il est possible, en la plaçant ailleurs, d'avoir une pile des retours plus étendue (par exemple pour des applications utilisant la récursivité, ou sauvegardant beaucoup de valeurs sur la pile des retours)...

RP@ --- **adr**

Donne la valeur courante du pointeur de la pile des retours. Par exemple, pour connaître la profondeur utilisée de la pile des retours :

```
RØ RP@ - 2/ 2- .  ( 2/ car une adresse s'écrit en deux octets et 2 - car la pile)
                  ( des retours contient toujours au moins deux adresses...)
```

Petit piège : si vous voulez définir un mot TAILLE-RETOURS, il faut enlever 1 pour tenir compte du mot lui-même !!!

```
: TAILLE-RETOURS ( --- n ) RØ RP@ - 2/ 3 - ;
```

Si vous utilisez TAILLE-RETOURS dans un mot, vous trouverez en général 1. Si ce mot est lui-même dans une autre mot, vous obtiendrez 2... A moins que vous l'ayez utilisé dans des boucles DO...LOOP !!!!...

S0 --- **adr**

Donne l'adresse initiale de la pile des données (base de la pile, qui en réalité est l'adresse la plus haute, puisque les piles s'étendent vers les adresses basses). SØ est la valeur de la variable SPØ. Le fait de mettre SØ en variable permet de reconfigurer le système.

SP! ---

Initialise le pointeur de la pile des données à la valeur contenue dans la variable SPØ.
Ce mot peut être utilisé pour "vider" la pile, ou après une reconfiguration du système (voir SPØ).

SP0 --- **adr**

Variable contenant l'adresse initiale du pointeur de la pile des données. En modifiant sa valeur (et en réinitialisant le pointeur courant de la pile des données par SP!), on peut donc reconfigurer le système. D'origine, la pile des données est limitée à 128 octets, ce qui suffit pour pratiquement toutes les applications. Il est possible, en la plaçant ailleurs, d'avoir une pile des données plus étendue (par exemple pour des applications utilisant la récursivité, ou sauvegardant beaucoup de valeurs sur la pile des données)...

SP@ --- **adr**

Donne la valeur courante du pointeur de la pile des données. Par exemple, pour connaître la profondeur utilisée de la pile des données, le mot DEPTH peut être défini par :

```
: DEPTH-1 ( --- n ) S0 SP@ - 2- 2/ ;
```

Le 2– provient de l'ajout de la valeur SØ sur la pile au moment où on utilise SP@)
En réalité, DEPTH est écrit en langage machine...

SQUISH n1 n2 --- n3

Opération inverse de HILO. SQUISH laisse sur la pile le nombre n3 dont l'octet de poids faible est l'octet de poids faible de n1, et dont l'octet de poids fort est l'octet de poids faible de n2.

```
HEX 12 45 SQUISH . 4512 OK
DECIMAL 12 45 SQUISH . 11532 OK ( car 11532=45x256+12)
```

Et même : HEX A212 3645 SQUISH . 4512 OK

SWAP n1 n2 --- n2 n1

Echange les deux nombres 16 bits du sommet de la pile.

```
12 13 .S      La pile affiche : 12 13
SWAP .S       La pile affiche : 13 12
SWAP .S
```

Nous voici revenus au point de départ.

SWAPDROP n1 n2 --- n2

Supprime le deuxième nombre 16 bits à partir du sommet de la pile. L'effet de SWAPDROP est le même que celui de SWAP suivi de DROP (c'est d'ailleurs exactement ainsi qu'il est construit).

```
12 13 14 SWAPDROP .S  La pile affiche : 12 14
```

Chapitre 2

Calculs arithmétiques et logiques

Mots relatifs aux calculs arithmétiques

Le FORTH standard ne calcule qu'avec des nombres entiers, 16 ou 32 bits (c'est-à-dire deux ou quatre octets). Par contre, il offre sur ces nombres une grande variété d'opérations, et travaille avec une très grande rapidité.
Pour la plupart des applications, cela n'est en aucun cas une limitation, et l'effort supplémentaire de programmation que nécessite cette absence des nombres "flottants" est largement compensée par la plus grande rapidité des opérations. Il est intéressant de noter à ce propos qu'une grande partie des applications qui fonctionnent en FORTH (ex : astronomie, contrôle de processus, robotique,...) sont des applications scientifiques comportant des calculs considérés d'habitude comme très complexes, même en utilisant des nombres "flottants" et des fonctions transcendantes (ex : Transformation de Fourier). Le coté interactif de FORTH, et la facilité de développement des applications est aussi pour beaucoup dans le choix de ce langage pour telle ou telle application.

Les nombres 32 bits sont entrés au clavier en mode interactif en incorporant un point parmi leurs chiffres (pas dans la réponse à un INPUT). Peu importe la position de ce point, il suffit qu'il y en ait un.
Par exemple, 3ØØØ. et 3.ØØØ représentent le même nombre 32 bits (de valeur 3ØØØ). La position du point décimal est mémorisée (pour le dernier nombre 32 bits entré) dans la variable DPL (voir ce mot, dans la section "formatage des entrées-sorties numériques", au chapitre 5), et peut être utilisée pour faire des calculs décimaux en virgule fixe (simulation de calculs avec des nombres "flottants").
Les nombres 32 bits sont représentés sur la pile comme deux nombres 16 bits, les 16 bits de poids fort étant sur le dessus de la pile. Pour utiliser comme nombre 16 bits une valeur 32 bits compris entre −32768 et 32767, il suffit d'enlever cette partie de poids fort par DROP. Cela est évident pour les nombres positifs, puisque leurs 16 bits de poids fort sont à Ø. En ce qui concerne les nombres négatifs, cela tient au fait qu'ils sont représentés en "complément à deux", de sorte que leurs 17 bits de poids fort sont à 1 (cette question dépasse l'objet de ce manuel. Si le sujet vous intéresse, il existe de nombreux ouvrages sur la représentation des nombres en mémoire dans un ordinateur).

Mots concernant les comparaisons numériques

Ces mots comparent des nombres (16 bits ou 32 bits) et laissent sur la pile une valeur de vérité (notée cond), qui vaudra toujours 1 si la comparaison amène un résultat "vrai", et Ø dans le cas contraire.

Mots concernant les opérations logiques

Ces mots sont souvent utilisés pour travailler sur les résultats de comparaisons, mais à la différence des mots de comparaison (numérique ou chaîne de caractères) ils ne laissent pas nécessairement Ø ou 1 sur la pile (lorsqu'on les utilise avec des nombres qui ne sont pas Ø ou 1). Ces mots réalisent les opérations booléennes fondamentales, mais les

mots AND, OR, et XOR le font bit à bit, sur les nombres 16 bits du sommet de la pile.
De sorte que par exemple on aura : `1 2 AND . 0 OK`
et cela bien que 1 et 2 soient tous deux compris comme étant la valeur booléenne "vrai" par les mots qui utilisent les conditions, comme IF, UNTIL, WHILE,...
L'expression ci-dessus pourrait donc se lire :
"vrai" et "vrai" = "faux"!!!
L'explication est simple, en binaire 1 s'écrit 0000000000000001 et 2 s'écrit 0000000000000010. Le "**et**" se faisant bit à bit, le résultat est 0000000000000000...
Il convient donc d'être prudent si on utilise AND ou XOR (les autres opérateurs ne posent pas ces problèmes) comme opérateurs logiques avec des valeurs numériques sur la pile qui ne sont pas 0 ou 1. Une bonne pratique est de n'utiliser dans ce cas AND et XOR que sur les résultats de comparaisons.

Ceci n'est pas une restriction ; il est parfois intéressant, pour réaliser des masquages de bits (par exemple lors d'utilisations graphiques) de pouvoir utiliser des "et" ou des "ou exclusif" bit à bit.
Les mots AND, OR, XOR peuvent être par ailleurs d'un grand intérêt pour gérer des "bit-maps" ou des tableaux compactés de valeurs booléennes (dans une variable 16 bits, on peut considérer que chacun des bits est une variable booléenne). Voir aussi le mot TOGGLE, dans le chapitre 8.

* n1 n2 --- n3

Multiplication de deux nombres entiers 16 bits. Résultat en 16 bits. Laisse sur la pile le produit n3 de n1 et n2.

```
33 5 * . 165 OK
```

Voir aussi 2* (multiplication rapide par 2), ainsi que D*, M*, U*, DN*, et UDN*.

*/ n1 n2 n3 --- n4

Opération combinée de multiplication-division.
Calcule le produit des nombres n1 et n2 (nombres entiers 16 bits), obtient un résultat qui est un nombre entier 32 bits, puis divise ce résultat par n3 (16 bits) et enfin dépose sur la pile le quotient entier n4 (16 bits).
Cette opération est plus sûre que la séquence FORTH :
n1 n2 * n3 /
En effet, l'opération * donne un résultat erroné si le produit est supérieur à 32767 (ou inférieur à −32768). Avec */, puisque le résultat de la multiplication est un nombre 32 bits, ce résultat sera exact. En fait, * utilise les opérations mixtes M* et M/.

```
1000 100 5*/. 20000 OK
```

1000 100 * 5 / donnerait un résultat incorrect car 100000 (résultat intermédiaire), n'est pas un nombre 16 bits. */ ne donne pas le reste de la division (voir */MOD).

*/MOD n1 n2 n3 --- n4 n5

Analogue à */ , à cela près que le résultat est ici :

n4 ..... le reste de la division de n1 × n2 par n3 (même signe que n1 × n2).

n5 ..... le quotient de cette division.

Comme pour */ le résultat intermédiaire (produit n1 × n2) est un nombre 32 bits (en fait, */ est défini par */MOD suivi de DROP).

```
500 600 7000 */MOD . . 42 6000 OK ( n1×n2=n3×n5+n4)
                                  ( quotient=42 reste=6000)
```

Attention à la convention de représentation de la pile, qui inverse l'ordre de l'écriture. On aurait ici :

```
500 600 7000 --- 6000 42
```

Par ailleurs, ici aussi l'opération 500 600 * 7000/MOD donnerait un résultat faux, car 500×600 est supérieur à 32767.

+ n1 n2 --- n3

Addition de deux nombres 16-bits, résultat en 16 bits.
n3 est la somme de n1 et n2.

```
21345 123 + . 21468 OK
32767 1 + . -32768 OK
```

Ce dernier résultat est incorrect, car 32768 n'est pas un nombre 16-bits signé...
Par contre : 32767 1 + U. 32768 OK (affichage 16-bits signé).
Voir aussi 1+ et 1− (additions "spécialisées") et D+.

+− n1 n2 --- n3

Mot système utilisé par ABS et NEGATE. Applique à n1 le signe de n2 (règle des signes). n3 vaut n1 ou −n1 , suivant que n2 est positif ou négatif.

```
23 -12 +- . -23 OK
-23 -45 +- . 23 OK
```

En fait, NEGATE et ABS sont définis par :

```
: NEGATE -1 +- ;   : ABS DUP +- ;
```

− n1 n2 --- n3

Soustraction de nombres 16 bits. n3 est la différence n1 − n2.

```
66 5 - . 61 OK
```

Voir aussi 1− (décrémentation) et D−.

/ n1 n2 --- n3

Division (entière) de nombres 16 bits, sans reste. n3 est le quotient entier de n1 par n2.

```
47 6 / . 7 OK     ( 47=6x7+5)
```

Voir aussi MOD et /MOD (pour obtenir aussi le reste). FORTH dispose de nombreux autres opérateurs de division :
2/ M/ M/MOD U/MOD.

/MOD **n1 n2 --- n3 n4**

Effectue la division entière de n1 par n2, et dépose sur la pile le reste n3 et le quotient n4. n3 a le même signe que n1.

```
47 6 /MOD . . 7 5 OK    ( n1=n2×n4+n3)
```

0< **n --- cond**

cond vaut 1 (VRAI) si n est négatif, et 0 si n est positif ou nul.

```
33 0< . 0 OK
-5 0< . 1 OK
33000 0< . 1 OK     ( 33000 est négatif !!!)
```

0= **n --- cond**

cond vaut 1 si n est nul, 0 si n n'est pas nul.

```
45 0= . 0 OK
0 0= . 1 OK
```

Noter que 0= transforme 1 (ou tout nombre non nul) en 0, et 0 en 1... Autrement dit, 0= est synonyme de NOT (voir mots de logique, au début de ce chapitre).

0> **n --- cond**

cond vaut 1 si n est strictement négatif, 0 si n est positif ou nul.

```
0 0> . 0 OK (0 n'est pas strictement positif)
35 0> . 1 OK
40000 0> . 1 OK (40000 est un nombre négatif !!!)
```

1+ **n --- n+1**
1- **n --- n-1**
2+ **n --- n+2**
2- **n --- n-2**
2* **n --- 2×n**
2/ **n --- n/2**

Opérateurs très souvent utilisés, écrits en langage machine pour une plus grande efficacité. Il s'agit en fait d'opérations spécialisées, où

l'un des opérandes est connu. Elles consistent à "ajouter ou soustraire 1 au nombre sur la pile" (incrémentation et décrémentation. Voir aussi 1+! et 1–! pour incrémenter ou décrémenter une variable), et "ajouter ou soustraire 2, multiplier ou diviser par 2".
Le gain en vitesse d'exécution (notamment pour la multiplication et la division par 2) est tel que pour multiplier un nombre par 64, par exemple, il est beaucoup plus rapide d'utiliser 6 fois l'opérateur 2*.

```
: TEST1 10000 0 DO 20 2* 2* 2* 2* 2* 2* DROP LOOP BELL ;
: TEST2 10000 0 DO 20 64 * DROP LOOP BELL ;
```

Chronométrez TEST1 et TEST2....

< n1 n2 --- cond

cond vaut 1 si n1 est inférieur à n2, 0 sinon.
Il s'agit d'une comparaison entre nombres 16 bits signés.

```
12 25 < . 1 OK
32767 32768 < . 0 OK      ( 32767 est le plus grand nombre positif ...)
```

Pour une comparaison entre nombres non signés, utiliser U<.

= n1 n2 --- cond

cond vaut 1 si n1 est égal à n2, 0 sinon.

```
12 3 4 * = . 1 OK
```

Autre exemple : compter les "A" tapés au clavier...

```
: TAPER-N-"A"    ( n ---)
        0                                   ( Initialise le compte des "A")
        BEGIN
            KEY DUP EMIT 65 = +             ( Un A de plus, le cas échéant)
            DDUP =                          ( A-t-on le compte demandé ??)
        UNTIL
        DDROP ;
```

33 TAPER-N-"A" vous laissera taper autant de caractères que vous voudrez, jusqu'à ce que vous ayez tapé 33 fois le caractère "A".

> n1 n2 --- cond

cond vaut 1 si n1 est supérieur à n2, Ø sinon.

```
12 25 > . Ø OK
32767 32768 > . 1 OK    ( 32768 est un nombre négatif)
```

Pour comparer des nombres non signés, vous pouvez définir U > par :

```
: U> ( n1 n2 --- cond ) DDUP = >R U< R> OR NOT ;
```

ABS n1 --- n2

n2 est la valeur absolue du nombre n1 (nombre 16 bits).

```
123 ABS . 123 OK
-23 ABS . 23 OK
```

AND n1 n2 --- n

réalise le "et" logique bit à bit.
Lorsque n1 et n2 valent Ø ou 1, le résultat n vaudra 1 lorsque les deux sont 1, et Ø si l'un au moins des deux nombres est Ø.

```
1 1 AND . 1 OK
1 Ø AND . Ø OK
Ø 1 AND . Ø OK
Ø Ø AND . Ø OK
```

Exemple d'utilisation : recherche d'un nom dans une liste... (Tant qu'on n'a pas fini et pas trouvé, continuer à chercher)
FINI et TROUVE étant des valeurs booléennes (variables valant Ø ou 1) définies et mises à jour par ailleurs, on aura par exemple :

```
... BEGIN
    FINI @ NOT
    TROUVE @ NOT AND
WHILE ...    ( chercher...)
REPEAT ...
```

Si n1 ou n2 ont des valeurs autres que Ø ou 1, le résultat est celui qu'on obtient en les écrivant en base deux, puis en réalisant la même opération pour chacun des chiffres.

```
2 BASE ! 1100 1010 AND . 1000 OK
```

C'est-à-dire : DECIMAL 12 10 AND . 8 OK
Exemple : mettre à Ø le bit 5 de l'octet situé à l'adresse adr... En mode direct :

```
adr C@ 2 BASE ! 11101111 DECIMAL AND adr C!
```

D* d1 d2 --- d3

Multiplie les nombres d1 et d2, et dépose sur la pile leur produit d3. (Multiplication 32 bits par 32 bits, résultat en 32 bits).

```
40000. -40000. D* D. -1600000000 OK    ( d3=d1 x d2)
```

Attention, si vous utilisez de très grands nombres, les nombres 32 bits ont aussi une limite supérieure. Par exemple, 50000. 60000. D* donne un résultat faux (le plus grand nombre positif 32 bits est 2147483647)...

D+ d1 d2 --- d3

Addition de deux nombres entiers 32 bits. Résultat en 32 bits.

```
1000000000. 1111111111. D+ D. 2111111111 OK
```

D+ – d1 d2 --- d3

La version 32 bits de + – (Règle des signes).
Si d2 est positif, d3 est égal à d1. Si d3 est négatif, d3 est l'opposé de d1.

D– d1 d2 --- d3

Soustraction de nombres 32 bits. d3 est la différence d1 – d2.

```
2000000000. 50000000.0 D- D. 1500000000 OK
```

D0= d --- cond

cond (nombre 16 bits) est vrai (égal à 1) si d est égal à 0 (nombre 32 bits), faux sinon.

```
123456. D0= . 0 OK
```

Par exemple, on peut définir un mot TOTAL avec la série d'instructions ci-après qui permet d'entrer au clavier une suite de nombres 32 bits, jusqu'à ce qu'on tape 0, et calculer leur somme.

```
: TOTAL    ( --- total)
      0.( Initialise le total)
      BEGIN CR ." Nombre : "*
         INPUT DDUP D0=  NOT      ( Lit un nombre et le teste)
      WHILE  D+                   ( Tant qu'il n'est pas nul, ajoute à Total)
      REPEAT
      DDROP CR ." ----------" CR ." Total : "* D. ;
```

Note : lorsqu'on exécute TOTAL, il n'y a pas à taper le point comme lorsqu'on tape un nombre double longueur en mode interactif. En effet, le mot INPUT interprète le nombre entré comme un nombre 32 bits (voir chapitre 9).

D< d1 d2 --- cond

cond (nombre 16 bits) vaut 1 si d1 (nombre 32 bits) est inférieur à d2, 0 sinon.

```
123456. 234567. D< . 1 OK
2.000000000 4.000000000 D< . 0 OK ( Comparaison signée... Voir DU< )
```

D= d1 d2 --- cond

cond (nombre 16 bits) vaut 1 si d1 (nombre 16 bits) est égal à d2, 0 sinon.

```
.10000000 2. D* 2000.0000 D= . 1 OK
```

DABS d1 --- d2

d2 est un nombre 32 bits positif, la valeur absolue du nombre 32 bits d1.

```
-132. DABS D. 132 OK
1.32 DABS D. 132 OK
```

DMAX d1 d2 --- d3
DMIN d1 d2 --- d3

DMAX laisse sur la pile le plus grand des deux nombres 32 bits d1 et d2, DMIN laisse le plus petit des deux.

```
100000. 2.000000   DMAX D. 2000000 OK
100000. 2.000000   DMIN D. 100000 OK
```

DN* d1 n --- d2

Multiplication mixte d'un nombre 32 bits par un nombre 16 bits. DN* multiplie le nombre 32 bits d1 par le nombre 16 bits n. Le résultat d2 (nombre 32 bits) reste sur la pile.

```
30000. 4 DN* D. 120000 OK
```

DNEGATE d --- −d

Change le signe du nombre 32 bits du sommet de la pile.

```
45.000 DNEGATE D. -45000 OK
-4500.0 DNEGATE D. 45000 OK
```

DU< ud1 ud2 --- cond

Comparaison de deux nombres 32 bits, interprétés comme non signés. cond vaut 1 si ud1 est inférieur à ud2, Ø dans le cas contraire.

```
2000000000. 4.000000000 DU< . 1 OK
```

Comparer ce résultat avec celui que donnerait D< ...

M* n1 n2 --- d

Multiplication de deux nombres 16 bits, résultat en 32 bits. A utiliser lorsque le produit n1 × n2 ne peut être (ou peut ne pas être) un nombre 16 bits.

```
30000 20000 M* D. 600000000 OK      ( d=n1×n2)
```

Voir aussi U* (analogue de M* en non signé).

M/ d n1 --- n2 n3

Division mixte avec reste. Les nombres 16 bits n2 et n3 sont respectivement le reste et le quotient de la division du nombre 32 bits d par le nombre 16 bits n1. Le reste n2 a le même signe que le nombre d.

```
-3000000. 170 M/ . . -17647 -10 OK    ( d=n1×n3+n2)
```

Le nom de cette commande n'est pas très satisfaisant (il s'agit d'une division avec reste, donc le nom devrait contenir le mot MOD. De sorte que M/MOD eût été plus adapté), mais il est standard.
M/ est en quelque sorte l'opération inverse de M* (en nombres entiers, l'opération inverse de la multiplication est la division avec reste !!!)

M/MOD ud1 un1 --- un2 ud2

Division mixte entre nombres non signés. M/MOD divise le nombre 32 bits non signé ud1 par le nombre 16 bits non signé un1, et laisse sur la pile le reste un2 (16 bits non signé) et le quotient (32 bits non signé).

```
90000000. 1900 M/MOD UD. U. 47368 800 OK ( ud1 = un1 x ud2 + un2)
```

MAX n1 n2 --- n3
MIN n1 n2 --- n3

MAX laisse sur la pile le plus grand des deux nombres n1 et n2, MIN laisse le plus petit des deux. Ces deux opérateurs sont souvent utilisés pour s'assurer que le nombre sur la pile est compris entre deux valeurs...

```
34 45 MIN . 34 OK
34 45 MAX . 45 OK
: LIT-LETTRE ( --- n) KEY 65 MAX 70 MIN ;
```

LIT-LETTRE laisse sur la pile un nombre qui ne peut être que le code ASCII de A, B, C, D, E, ou F (par exemple pour choisir une option dans un menu). Si on a tapé par exemple un chiffre, le résultat sur la pile sera 65 (code de A).

MOD n1 n2 --- n3

MOD laisse sur la pile le reste n3 de la division de n1 par n2. n3 a le même signe que n1. L'origine de ce nom est le mot mathématique "modulo" (n3 est égal à n1 modulo n2).

```
45 7 MOD . 3 OK      (Si on divise 45 par 7, il restera 3)
```

MOD n'a pas son équivalent pour la division de nombres 32 bits ou pour les opérations mixtes (16 – 32 bits), mais il est possible d'utiliser M/ (ou M/MOD ou encore U/MOD) suivi de DROP (ou DDROP, suivant que le quotient est un nombre 16 ou 32 bits).

NEGATE n --- -n

Change le nombre du dessus de la pile en son opposé.

```
45 NEGATE . -45 OK
-45 NEGATE . 45 OK
```

NOT n1 --- n2

Synonyme de Ø=. n2 vaut Ø si n1 n'est pas nul, 1 si n1 est nul. NOT ne travaille pas bit à bit (pour réaliser bit à bit l'opération correspondant à NOT, vous pouvez utiliser XOR,... ou TOGGLE pour travailler directement en mémoire).

```
12 NOT . Ø OK
Ø NOT . 1 OK
```

Souvent utilisé avec WHILE dans des formulations du genre : "Tant qu'on n'a pas fini, on continue" (une des phrases clés de la programmation).

```
BEGIN FINI NOT WHILE ...(continuer)... REPEAT
```

Pour être sûr que la pile contient une valeur booléenne (Ø ou 1), on utilise souvent la séquence d'instructions "Ø= NOT" (ou NOT NOT)...

```
: BOOL ( n1 --- n2) NOT NOT ;
123 BOOL . 1 OK
2 BOOL . 1 OK
Ø BOOL . Ø OK
```

OR n1 n2 --- n

Réalise le "ou" logique bit à bit. Lorsque n1 et n2 valent Ø ou 1, le résultat n vaudra Ø lorsque les deux sont Ø, et 1 si l'un au moins des deux nombres est 1.

```
1 1 OR . 1 OK
1 Ø OR . 1 OK
Ø 1 OR . 1 OK
Ø Ø OR . Ø OK
```

Exemple d'utilisation : recherche d'un nom dans une liste...
(Tant qu'on n'a pas fini et pas trouvé, continuer à chercher)
FINI et TROUVE étant des valeurs booléennes (variables valant Ø

ou 1) définies et mises à jour par ailleurs, on aura par exemple :

```
... BEGIN
    FINI @     ( Arrêter si on a)
    TROUVE @ OR NOT     ( fini ou trouvé ...)
  WHILE      ( chercher...)
  REPEAT ...
```

Si n1 ou n2 ont des valeurs autres que Ø ou 1, le résultat est celui qu'on obtient en les écrivant en base deux, puis en réalisant la même opération pour chacun des chiffres.

```
2 BASE ! 1100 1010 OR . 1110 OK
```

C'est-à-dire : DECIMAL 12 10 OR . 14 OK

Exemple : mettre à 1 le bit 5 de l'octet situé à l'adresse adr. (Supposons adr donnée en décimal...)

En mode direct : adr C@ 2 BASE ! ØØ1ØØØØ DECIMAL OR adr C!

S– >D n --- d

Transforme le nombre 16 bits n en nombre 32 bits. Si n est un nombre positif, l'action de S– > D est simplement d'empiler le nombre Ø (et vous pouvez simplement utiliser Ø au lieu de S– > D). Si n est négatif, S– > D étend le signe de n en empilant le nombre binaire 1111111111111111.

```
345 S- >D D. 345 OK
345 Ø D. 345 OK
 -45 S- >D D. -45 OK
-45 Ø D. 65491 OK    ( Résultat faux, car le signe n'a pas été étendu)
```

U* un1 un2 --- ud

Multiplication entre nombres 16 bits non signés, résultat en 32 bits.

```
40000 50000 U* D. 2000000000 OK    ( ud=un1 x un2)
```

Pensez à ce genre d'opérations... FORTH, utilisant des nombres entiers, fait un grand usage des nombres non signés...

U/MOD ud un1 --- un2 un3

Division avec reste non signée d'un nombre 32 bits ud par le nombre 16 bits un1. Le reste un2 et le quotient un3 sont des nombres 16 bits non signés.

```
2500000000. 45000 U/MOD U. U. 55555 25000 OK
3000000000. 45000 U/MOD U. U. 65535 65535 OK
```

Le dernier résultat est évidemment faux, le quotient devrait être 66666. Mais 66666 n'est pas un nombre 16 bits, même non signé...

U< un1 un2 --- cond

Comparaison de deux nombres 16 bits, interprétés comme non signés. cond vaut 1 si un1 est inférieur à un2, 0 dans le cas contraire.

```
32767 32768 U< . 1 OK
65000 66000 U< . 0 OK
```

(Attention, en non signé, le nombre qui suit 65535 est 0 ...)

UDN* ud1 un --- ud2

Multiplication mixte d'un nombre 32 bits non signé (ud1) par un nombre 16 bits non signé (un).
ud2 est le produit non signé de ud1 et un.

```
400000. 50 UDN* UD. 20000000 OK
```

Ici encore, attention à la taille des nombres qui interviennent. Par exemple, le produit de 2000000000 par 3000 ne peut être un nombre 32 bits non signé, et le résultat sera faux.

XOR n1 n2 --- n

Réalise le "ou exclusif" bit à bit de n1 et n2. Lorsque n1 et n2 valent 0 ou 1, le résultat n vaudra 0 lorsque les deux sont égaux, et 1 s'ils sont différents.

```
1 1 XOR . 0 OK
1 0 XOR . 1 OK
0 1 XOR . 1 OK
0 0 XOR . 0 OK
```

Si n1 ou n2 ont des valeurs autres que Ø ou 1, le résultat est celui qu'on obtient en les écrivant en base deux, puis en réalisant la même opération pour chacun des chiffres.

```
2 BASE ! 1100 1010 XOR . 110 OK
```

C'est-à-dire : `DECIMAL 12 10 XOR . 6 OK`

Chapitre 3

Structures de contrôle

Ces mots constituent l'un des points forts de FORTH. Toutes les structures définies en programmation structurée existent en FORTH, et même pour certaines d'entre elles sont plus puissantes ici que celles qu'on utilise usuellement en algorithmique.
La conception même de FORTH exclut l'idée d'un quelconque "GOTO", encore que les mots BRANCH et ØBRANCH ne soient en fait que des "GOTO conditionnels et relatifs". Les mots de structuration du FORTH peuvent se diviser en plusieurs catégories.

Structures conditionnelles :

— Condition et alternative (Si, Alors, Sinon): IF ELSE THEN
— Sélection (Selon, Suivant): CASE et ses dérivés OF, ENDOF, ENDCASE
GODO (et THEN)

Structures répétitives :

— Boucles indicées
(boucle "Pour") : DO LOOP +LOOP /LOOP I J
— Boucles infinies ou s'arrêtant sur une condition : (commencent par BEGIN)

Répéter ... jusqu'à...	BEGIN UNTIL
Tant que... fin-tant-que	BEGIN WHILE REPEAT
Répéter indéfiniment	BEGIN AGAIN

Sortie exceptionnelle de boucle :

EXIT LEAVE ABORT

Mots système :
(utilisés pour construire les mots ci-dessus).

(+LOOP) (/LOOP) (DO) (GODO) (LOOP)
(OF) ØBRANCH BRANCH

Ces mots ne sont pas utilisables en mode interactif, et doivent être employés à l'intérieur de définitions par :...;
Les mots système peuvent être utilisés par les programmeurs expérimentés pour définir de nouvelles structures de contrôle, mais il est raisonnable de les ignorer dans l'écriture des applications au niveau le plus élevé. Cela est en particulier indispensable si on tient à la portabilité des applications.
En effet, les mots IF, ELSE, THEN, DO, LOOP, BEGIN, WHILE, REPEAT, UNTIL,... sont standards et existent dans tous les FORTH qui respectent le standard FORTH-79. Par contre, les primitives (entre parenthèses) sont liées au système, et ne feront pas nécessairement partie d'un autre FORTH, même standard.

(+LOOP) (/LOOP) (DO) (GODO) (LOOP) (OF)

Mots système, en général écrits en langage machine, et compilés par les mots : +LOOP /LOOP DO GODO LOOP et OF.

+LOOP **n** --- Immédiat, Seulement dans :...;

A la fin d'une boucle DO, permet d'utiliser un incrément signé n. L'indice de la boucle est incrémenté de n (décrémenté si n < Ø), et l'exécution se continue par le mot qui suit DO jusqu'à ce que l'indice soit supérieur ou égal à n (si n est positif), ou inférieur à n (si n est négatif). La boucle DO ... +LOOP est tout à fait analogue à la boucle FOR ...STEP ...NEXT du BASIC.
1 +LOOP est équivalent à LOOP (en moins rapide).

Voir aussi DO LOOP et /LOOP.

```
: NBRES-PAIRS  ( n1 n2 ---) DO I . 2 +LOOP ;
11 0 NBRES-PAIRS 0 2 4 6 8 10 OK
```

Avec un incrément négatif :

```
: DEGRINGOLADE  (  ---) 49 60 DO I NOTE -1 +LOOP ;
                                    ( Joue une gamme chromatique descendante)
```

/LOOP **un ---** Immédiat, Seulement dans :...;

Mot de fin de boucle DO. /LOOP est analogue à +LOOP, à ceci près que l'incrément et l'indice sont des nombres non signés, et que la logique utilisée est la logique non signée.

```
: T1 40000 30000 DO I U. LOOP ;    (ne fonctionne pas, n'affiche que 30000).
```

En effet, en arithmétique signée, 40000 est négatif (donc inférieur à 30000).

Utiliser TT1, défini comme suit :

```
: TT1 40000 30000 DO I U. 1 /LOOP ;
```

De même : T2 (ci-dessous) ne fonctionne pas...

```
: T2 40000 30000 DO I U. 5 +LOOP ;
                                    ( même raison que pour T1)
```

Utiliser TT2 :

```
: TT2 40000 30000 DO I U. 5 /LOOP ;
```

Par contre, il ne faut pas oublier que /LOOP interprète les nombres comme étant non signés, et que si on lui donne un incrément négatif les résultats peuvent être curieux...

```
: T3 -5000 30000 DO I . -10000 /LOOP ;
```

affiche les nombres :

```
30000 20000 10000 0 -10000 -20000 -30000 25536 15536 5536
```

Explication : la boucle s'arrête lorsque l'indice est supérieur à −5 en non signé. Et en non signé, les nombres ci-dessus s'écrivent 30000, 20000, 10000, 0, 55536, 45536, 35536, 25536, 15536, et 5536 (tous inférieurs à −5000, c'est-à-dire à 60536). Le suivant serait 5536 − 10000, donc −4464, qui s'écrit 61072 en non signé...

Moralité : n'utilisez /LOOP qu'avec un incrément positif et des valeurs limites positives (de 0 à 65535), ou alors faites-le en connaissance de cause...

ØBRANCH cond ---

Mot système compilé par IF, UNTIL, WHILE pour réaliser les branchements conditionnels. Il s'agit de branchements relatifs, l'offset utilisé étant le paramètre qui suit ØBRANCH dans la définition du mot en cours.

?PAIRS n1 n2 ---

Mot système utilisé lors de la compilation, pour vérifier la correction de la syntaxe. ?PAIRS affiche le message "CD ERROR" si n1 n'est pas égal à n2...

AGAIN --- Immédiat, Seulement dans :...;

Effectue un saut inconditionnel au début d'une boucle BEGIN...AGAIN . Une telle boucle est infinie, et ne peut être interrompue que par un redémarrage du système.

```
: SCIE BEGIN BELL AGAIN ;
```

Seul le bouton d'initialisation (ou la prise de courant !...) pourra vous tirer d'affaire...
L'interpréteur FORTH utilise cette boucle dès la mise sous tension : votre TO-7 passe son temps, lorsque vous faites du FORTH, à exécuter sans cesse une boucle consistant à lire le flot d'entrée, l'interpréter, le traiter, et recommencer...

BACK adr ---

Mot système utilisé par UNTIL, REPEAT,... pour calculer les offsets (adresse courante - adr) pour les branchements relatifs (lors de l'emploi de BRANCH ou de ØBRANCH).

Mot employé pour réaliser des boucles sans compteur. BEGIN permet trois constructions :

```
BEGIN < actions > AGAIN
      REPETER INDEFINIMENT < actions >
BEGIN < actions > cond UNTIL
      REPETER < actions > JUSQU'A < cond = 'vrai' >
BEGIN < actions1 > cond WHILE < actions > REPEAT
```

Cette dernière construction est la forme la plus puissante, mais la plus complexe à saisir. L'idée fondamentale est :

```
TANT-QUE < cond = 'vrai' > FAIRE < actions >
```

Cette idée est réalisée par la structure :

```
BEGIN cond WHILE < actions > REPEAT.
```

En fait, cond est une valeur de vérité, qui est le résultat (sur la pile) d'une série d'actions, < actions1 > , qui peut être plus ou moins complexe (voir exemple ci-après).
La structure avec WHILE est la plus puissante, car elle peut remplacer les autres dans tous les cas. Elle peut même remplacer IF..THEN !!

Dans la première version, < actions > est exécuté indéfiniment, et la boucle ne peut être arrêtée que par EXIT ou ABORT. Il est donc raisonnable que AGAIN soit le dernier mot d'une définition, juste avant le point-virgule.

```
: TEST-AGAIN  BEGIN KEY DUP 2 = IF EXIT THEN
                    7 MOD FRAME AGAIN
              BELL ;
```

TEST-AGAIN change la couleur du tour de l'écran en fonction des touches tapées, et s'arrête lorsqu'on appuie sur la touche STOP. Le mot BELL ne peut s'exécuter, même après appui sur < STOP > , car EXIT force la fin de l'exécution de TEST-AGAIN. Voir ci-dessous le mot TEST-WHILE, qui réalise la même boucle de manière plus simple et plus facile à lire.
Si on omet la séquence "DUP 2 = IF EXIT THEN", le seul moyen d'arrêter l'exécution de TEST-AGAIN sera le bouton de réinitialisation.

Dans la deuxième version, < actions > est exécuté jusqu'à ce que cond soit vrai.

```
: TEST-UNTIL  BEGIN KEY DUP 7 MOD FRAME 2 =
              UNTIL
              BELL ;
```

TEST-UNTIL réalise à peu près la même action que TEST-AGAIN, jusqu'à ce qu'on tape la touche STOP. Il fait alors un bip, signalant la fin de la boucle. Noter qu'à la différence de TEST-AGAIN, l'action est toujours exécutée au moins une fois (cas où la première touche tapée est la touche STOP), et qu'après l'appui sur STOP le tour de l'écran change une dernière fois de couleur. Après l'exécution de TEST-UNTIL, le tour de l'écran est toujours vert (couleur 2).

Dans la troisième version, < actions1 > est exécuté, et laisse une valeur de vérité sur la pile.
Si cette valeur vaut 1 (ou tout autre nombre non nul), < actions2 > est exécuté, et on revient à < actions1 > . Sinon, c'est le mot qui suit REPEAT qui est exécuté (fin de la boucle).

```
: TEST-WHILE  BEGIN KEY DUP 2 = NOT
              WHILE 7 MOD FRAME REPEAT
              BELL ;
```

L'action de TEST-WHILE est analogue à celle de TEST-AGAIN, à cela près que BELL est maintenant exécuté, et que la boucle a une fin. A la différence de TEST-UNTIL, la touche STOP n'est pas prise en compte pour la modification de la couleur, et si la première touche enfoncée lors de l'exécution est la touche STOP la couleur ne change pas.

Dans une boucle BEGIN...UNTIL, < actions > est toujours exécuté au moins une fois (test après l'action), alors que dans une boucle BEGIN...WHILE...REPEAT, < actions > peut très bien ne jamais être exécuté, si la condition n'est jamais réalisée. Peu de langages disposent d'une structure de contrôle aussi souple et puissante que la structure BEGIN...WHILE...REPEAT. Les équivalences suivantes donnent une idée de l'universalité de WHILE :

```
BEGIN XX AGAIN    équivaut à   BEGIN XX UNTIL
                  ou à         BEGIN 1 WHILE XX REPEAT
BEGIN YY UNTIL    équivaut à   BEGIN YY NOT WHILE REPEAT
                  ou à         YY BEGIN NOT WHILE YY REPEAT
```

Et même :

```
XX IF YY THEN     équivaut à   XX BEGIN WHILE YY Ø REPEAT
```

BRANCH ---

Mot système compilé par ELSE, AGAIN, REPEAT pour réaliser les branchements inconditionnels (comme pour BRANCH, branchements relatifs).

CASE **n** --- Immédiat, Seulement dans :...;

Une structure de contrôle évoluée, utilisant un sélecteur numérique. On l'utilise de la manière suivante: ...

```
n CASE
  n1 OF < actions1 > ENDOF
  n2 OF < actions2 > ENDOF
        ...
  np OF < actionsp > ENDOF
        < autres actions >
  ENDCASE ...
```

Lors de l'exécution, le nombre n qui figure sur la pile sert de sélecteur : si n est égal à n1, alors n et n1 sont enlevés de la pile, < actions1 > est exécuté, et l'exécution se poursuit par le mot qui suit ENDCASE. Si n est différent de n1, alors n1 est dépilé, n est comparé à n2, et ainsi de suite... Si n n'est égal à aucun des nombres n1, n2, ..., np, n reste sur la pile, < autres actions > est exécuté, et ENDCASE l'enlève de la pile avant de passer à la suite.

```
: TAPETOUCHE ( --- ) ." Tape un A "
  KEY CASE
        65 OF CR ." Tu as tapé un A "  2Ø ENDOF
        97 OF CR ." Tu as tapé un a "  1Ø ENDOF
        CR ." Boffff !!!..." Ø SWAP ( Ø sera dépilé par ENDCASE)
      ENDCASE
  CR CR ." Note :" 4 .R ." sur 2Ø" ;
```

DO **n1 n2** --- Immédiat, Seulement dans :...;

Ne s'utilise qu'à l'intérieur d'une définition par : ... ; pour réaliser une boucle indicée. Trois formes sont possibles :

n1 n2 DO < actions > LOOP ...

Forme habituelle. L'indice de boucle commence à la valeur n2 (ATTENTION A L'ORDRE), et est incrémenté de 1 par LOOP.

< actions > est exécuté jusqu'à ce que l'indice soit SUPERIEUR OU EGAL à n2. (Il n'y a pas d'action pour indice = n2)

```
: TEST 10 0 DO I . LOOP ;
TEST 0 1 2 3 4 5 6 7 8 9 OK
```

Il appartient au programmeur d'augmenter de 1 l'indice maximum de la boucle (prendre 97 pour aller jusqu'à 96).

n1 n2 DO < actions > p +LOOP ...

Analogue au STEP de BASIC. +LOOP ajoute p à l'indice à chaque passage. Même condition d'arrêt si p est positif (n1 > n2), mais si n est négatif la condition d'arrêt est que l'indice soit STRICTEMENT INFERIEUR à n1.

```
: TEST+ 20 2 DO I . 5 +LOOP ;
TEST+ 2 7 12 17 OK
: TEST- 0 20 DO I . -3 +LOOP ;
TEST- 20 17 14 11 8 5 2 OK
```

n1 n2 DO < actions > un /LOOP ...

Incrément non signé. Permet de réaliser des boucles dont l'indice va jusqu'à 65535, sans problème de comparaison... Par exemple, une boucle du type 32000 33000 DO...LOOP ne fonctionnerait pas car en nombres signés, 33000 est inférieur à 32000. Par contre, en remplacant LOOP par 1 /LOOP on n'aura pas de problème. Si vous avez de la patience :

```
DECIMAL : TEST/ 65535 0 DO I U. 128 /LOOP ;
        HEX TEST/ 0 100 200 ... FF00 OK
```

Notes :

A l'intérieur d'une boucle, on peut obtenir la valeur courante de l'indice en utilisant le mot I (ou R@).

Lors du début de la boucle, DO dépose sur la pile des retours la valeur extrême de l'indice, et la valeur de l'indice départ (d'où le R@ ci-dessus). Le mot LOOP incrémente la valeur de l'indice, puis teste la fin de la boucle en comparant les deux valeurs du sommet de la pile des retours. Les mots /LOOP et +LOOP ont des actions analogues à LOOP.

Il est donc fondamental de ne pas modifier les valeurs du sommet de la pile des retours à l'intérieur d'une boucle DO. Si on a par exemple utilisé > R dans la boucle, il est indispensable de restaurer l'état de

la pile des retours avant LOOP, en général par le mot R > .
Les boucles DO... peuvent être imbriquées, la seule limitation est la taille de la pile des retours. On ne peut donc pas imbriquer plus de 31 boucles DO (quatre octets nécessaires par boucle, sans compter les adresses de retour des autres mots), mais ceci n'est déjà pas si mal. Dans le cas de boucles imbriquées, on peut obtenir les indices des deux boucles les plus internes par les mots I et J (I donnant l'indice de la boucle la plus interne...).

ELSE Immédiat, Seulement dans :...;

Utilisé uniquement dans une définition par : ... ;
Voir IF.

ENDCASE
ENDOF Immédiats, Seulement dans :...;

Voir CASE

EXIT --- Seulement dans :...;

EXIT interrompt l'exécution du mot en cours, et retourne au mot qui l'a appelé (éventuellement à l'interpréteur).

```
: VROOM BEGIN 7 EMIT
            PTRIG IF EXIT THEN
        AGAIN ;
```

est équivalent à

```
: BZZZ  BEGIN 7 EMIT
            PTRIG
        UNTIL ;
```

N'utilisez jamais EXIT pour sortir d'une boucle DO...
FORTH interpréterait l'indice de la boucle (au sommet de la pile des retours) comme adresse du mot appelant. Il est concevable d'utiliser la séquence "R > R > DDROP EXIT", mais le plus simple est de se servir du mot LEAVE.

S'utilise sous la forme :

```
: XX Avant ( n ) GODO motØ mot1 mot2 ... motx THEN suite ;
```

XX exécutera le mot "Avant" (qui laisse sur la pile la valeur n), puis motn, puis les mots qui suivent THEN (Le mot "suite" dans cet exemple).
Si n est inférieur ou égal à Ø, motØ est exécuté.
Si n est égal à 1, mot1 est exécuté, et ainsi de suite...
Si n est supérieur ou égal à x, alors motx sera exécuté.
Cette structure ressemble beaucoup au "ON N GOSUB" du BASIC, ou encore au CASE de FORTH ou de PASCAL. Mais contrairement à CASE (de FORTH), l'action à exécuter doit ici consister en un nom de mot FORTH ; en outre, les valeurs du sélecteur sont nécessairement les nombres Ø, 1, 2, etc. (à partir de Ø).
Exemple : menu...

```
: OPT1 ....;    ( Actions à exécuter si on choisit l'option 1)
: OPT2 ....;    ( Actions à exécuter si on choisit l'option 2)
: OPT3 ....;    ( Actions à exécuter si on choisit l'option 3)
: OPT4 ....;    ( Actions à exécuter si on choisit l'option 4)
```

Puis : MENU

```
BEGIN
  AFFICHE-TEXTE-MENU
  KEY 48 - DUP                          ( Le code ASCII de "Ø" est 48)
  GODO  BELL                            ( Code de la touche inférieur à 49)
        OPT1 OPT2 OPT3 OPT4                        ( Options valides)
        BELL                                ( Codes trop grands ...)
  THEN  Ø =
UNTIL ;                                 ( Arrêt si on a tapé la touche "Ø" )
```

Ou encore, pour faire un menu où le choix s'effectue grâce au crayon optique, après avoir défini des zones de pointage par les mots PEN ou BOXPEN... (voir ces mots).

```
: MENU
            BEGIN  AFFICHE-TEXTE-MENU  DEFINIT-ZONES
              ONPEN                ( Laisse le numéro de la zone pointée)
              GODO BELL OPT1 OPT2 OPT3 OPT4 EXIT THEN
            AGAIN ;
```

Noter qu'ici l'option 5 (EXIT) rend la main à l'interpréteur si le mot MENU est le mot "final" de l'application en cours. Une autre possibilité est le mot ABORT, qui rend la main dans tous les cas.

I --- n

Dans une boucle DO...LOOP, I copie sur la pile des données l'indice de boucle (qui se trouve sur la pile des retours).

```
: TEST-I  10 0 DO KEY DROP CR ." Indice de boucle : "  I . LOOP ;
```

IF cond --- Immédiat, Seulement dans :...;

Instruction de sélection. Deux constructions sont possibles :

cond IF < action > THEN

ou cond IF < action > ELSE < action1 > THEN

Si la condition est réalisée (si cond n'est pas nul), c'est < action > qui est exécuté, et < action1 > est ignoré. Si la condition n'est pas réalisée, alors dans le deuxième cas < action > sera ignoré, et < action1 > exécuté. Dans le premier cas, l'exécution se continue par le mot qui suit THEN.

```
: GAG      PLUIE IF PARAPLUIE PRENDRE THEN SORTIR ;
: GAGAG    PLUIE
           IF PARAPLUIE
           ELSE OMBRELLE THEN
           PRENDRE SORTIR ;
```

Dans le premier cas, s'il ne pleut pas, aucune action ne sera effectuée avant de sortir. Dans le deuxième, on prendra soit une ombrelle, soit un parapluie suivant qu'il pleut ou non.

J --- n

Dans des boucles DO...LOOP imbriquées, J donne la valeur de l'indice de la boucle externe (en appelant première la plus interne).

```
: TESTJ 10 0 DO CR I . CR 4 0
                DO ." //" J 2 .R ." - "  I 2 .R LOOP
                KEY DROP LOOP CR ;
```

Voir DO, LOOP, I.

LEAVE --- Seulement dans :...;

Force la sortie anticipée d'une boucle DO. L'action de LEAVE est de modifier la valeur de l'indice terminal, en lui donnant la valeur de l'indice de boucle.

Par conséquent, l'exécution ultérieure du mot LOOP (ou +LOOP ou /LOOP) terminera la boucle.
Attention, la boucle ne s'arrête pas immédiatement, la sortie se fait en fin de boucle. En particulier, les actions qui suivent LEAVE à l'intérieur de la boucle seront exécutées.

```
: ESSAI-LEAVE
        100 0 DO
                KEY DUP 2 = IF LEAVE THEN
                DUP CR EMIT 3 .R BELL LOOP ;
```

La boucle est prévue pour s'exécuter une fois au maximum, mais s'arrête si on tape la touche STOP. Après avoir tapé le < STOP > qui provoque la fin de boucle, vous entendrez un coup de sonnette, et vous verrez s'afficher le code ASCII (2) de la touche < STOP > (le BELL qui suit LEAVE est exécuté avant le LOOP final, ainsi que l'affichage de la touche).
LEAVE permet d'utiliser la boucle DO dans des situations analogues à celles où on utilise le plus souvent BEGIN...UNTIL, à cela près que le nombre de répétitions est limité.

LOOP ---

Mot de terminaison de boucle DO. L'action de LOOP est d'incrémenter l'indice de boucle, puis de le comparer à l'indice maximum de la boucle. Si l'indice est inférieur à l'indice de fin, l'exécution se continue par le mot qui suit DO (la boucle continue). Dans le cas contraire LOOP nettoie la pile des retours (où sont empilés les indices de la boucle) et l'exécution se continue par le mot qui suit LOOP.

```
: VOIR-LOOP ( n1 n2 --- )
        1+ OVER DO
                KEY DROP                        ( Pour avoir le temps de voir)
                I OVER = IF ." Action de DO : "
                        ELSE ." Action de LOOP : "  THEN
                .S                                      ( Affiche les piles)
                LOOP
        ." Action finale de LOOP : "* .S DROP ;
```

L'intérêt de cette exemple est de surveiller l'évolution de la pile des retours. La pile des données contiendra toujours le nombre n1 (indice de départ).
Voir aussi les mots DO, +LOOP et /LOOP.

OF --- Immédiat, Seulement dans : ,

Voir CASE. OF marque le début du traitement d'une des éventualités prévues dans le CASE.

REPEAT --- Immédiat, Seulement dans :...;

Voir BEGIN.
REPEAT marque la fin d'une boucle BEGIN...WHILE...REPEAT, et se traduit à l'exécution par un branchement inconditionnel au mot qui suit BEGIN.

THEN --- Immédiat, Seulement dans :...;

Marque la fin d'une structure commencée par IF ou GODO. Voir ces mots.
Les structures concernées sont :
IF...THEN IF...ELSE...THEN GODO...THEN

UNTIL cond --- Immédiat, Seulement dans :...;

Voir BEGIN.
UNTIL marque la fin d'une boucle BEGIN...UNTIL, et se traduit par un branchement conditionnel au mot qui suit BEGIN (lorsque cond est "faux"). Si cond est "vrai", l'exécution se poursuit en séquence, par le mot qui suit UNTIL.

WHILE cond --- Immédiat, Seulement dans :...;

Voir BEGIN.
WHILE est le mot qui gère la boucle BEGIN...WHILE...REPEAT, et se traduit par un branchement conditionnel. Si cond est "faux", le branchement se fait sur le mot qui suit REPEAT, sinon l'exécution se poursuit en séquence par le mot qui suit WHILE.
WHILE donne une structure particulièrement puissante, tout à fait adaptée au traitement séquentiel (files, fichiers séquentiels), en particulier dans le cas fréquent où le dernier élément est un marqueur

de fin de traitement, et justifie d'un traitement spécial. L'utilisation type est la suivante :

```
... BEGIN  Obtenir l'élément à traiter
           Doit-on le traiter (est-ce le dernier ?)
   WHILE ..Traitement ...
   REPEAT ...
```

Exemple : le mot TOTAL calcule la somme d'une suite de nombres positifs 32 bits entrés au clavier, terminée par -1

```
: TOTAL
        0. ( Initialise le total. Attention au point indiquant un nombre 32 bits)
        BEGIN CR ." Nombre à ajouter : "  INPUT
              DDUP -1. D= NOT
        WHILE D+
        REPEAT
        DDROP CR CR ." Total : "  UD. ;
```

Chapitre 4

Définition de nouveaux mots Compilation et interprétation

Il convient avant d'aborder ce chapitre, de parler de ce qu'est la compilation.
Le mot "compiler" peut avoir en FORTH plusieurs significations voisines, mais n'a pas le sens que l'on lui attribue en général dans une expression comme "un langage compilé"...
En principe, l'action de "compiler une définition" consiste à reconnaître les mots qui la composent, et à mettre leurs adresses d'exécution dans le corps de la définition (dans son pf. Voir le lexique en annexe G, et l'annexe C pour la structure d'un mot FORTH), de sorte qu'un appel ultérieur de cette définition exécutera ces mots... Cela, c'est le premier sens, c'est celui qui concerne les mots ":" et ";" (sans les guillemets). C'est à ce fonctionnement qu'on fera allusion dans ce qui suit lors qu'on utilisera les expressions "mode compilation" ou "mode exécution".
Toutefois, la tradition, en FORTH, est d'appeler aussi "compilation" le fait de définir des nouveaux mots, et plus généralement d'ajouter des choses (restons vagues... Voir par exemple "ALLOT" ou ",") au dictionnaire... En ce sens, des mots comme CONSTANT, ARRAY, ou VARIABLE (entre autres...) sont des mots qui réalisent des compilations.
Mais ils ne positionnent pas le "mode compilation" (ils n'agissent pas sur la variable STATE).

Conclusion : si on parle de "Mode compilation", pas d'ambiguïté, cela a un sens très précis. (Voir par exemple la variable STATE et le mot INTERPRET...)
Si par contre on parle simplement de "Compilation", le sens dépend du contexte. Il s'agit simplement d'actions qui modifient le dictionnaire (en le faisant grossir).

On appelle souvent "Mots de définition" les mots qui peuvent définir d'autres mots, comme VARIABLE ou ARRAY... , et "Mots de compilation" les mots qui ont une action au moment de la compilation (entre ":" et ";"), comme IF, BEGIN,...
Signalons aussi que d'autres mots agissent sur le dictionnaire sans être d'un de ces deux types... Citons par exemple "ALLOT", "," (virgule), "C,", FORGET...
Nous n'avons gardé dans ce chapitre que les mots explicitement utilisés lors de la création de nouveaux mots. Les autres (comme FORGET) figurent dans le chapitre 8.
Dans le présent chapitre, vous trouverez aussi deux mots tout à fait particuliers, deux mots sans lesquels FORTH ne serait pas FORTH, deux mots qui donnent à ce langage toute sa puissance et sa souplesse... Il s'agit des mots CREATE et DOES > , qui permettent de définir des mots qui eux-mêmes définiront des mots (mots de définition). CREATE et DOES > sont surtout utilisés pour définir de nouvelles structures de données... VARIABLE, CONSTANT, ARRAY, CARRAY, TABLE, CTABLE, STRING, ARRAY$, sont autant d'exemples de mots qu'on peut définir grâce à ces deux petits mots... On peut bien sûr passer sa vie sans jamais utiliser ces mots, mais il serait dommage d'ignorer leur existence.
Tel autre langage dispose-t-il d'une structure de données qui vous parait particulièrement adaptée à votre problème ?
En FORTH, c'est tout simple, il vous suffit de fabriquer vous-même cette structure. Pour plus de détails, voir CREATE et DOES > ...

Les descriptions de ce chapitre supposent que vous connaissez la structure des mots du dictionnaire, et les termes qui s'y réfèrent (comme cf, lf, pf, cfa, etc.). Vous trouverez une explication sommaire des principaux termes employés dans le lexique (annexe G), et des précisions sur la structure du dictionnaire à l'annexe C.

ICSP ---

Mot système, dont l'action est de sauvegarder le pointeur de la pile des données dans la variable CSP. Utilisé en début de compilation, par le mot ":".
En fin de compilation, si tout s'est bien passé, la pile doit être revenue dans son état antérieur, de sorte que la comparaison du contenu de CSP avec le pointeur de pile donne une vérification du bon fonctionnement de la compilation (voir le mot ?CSP). Cela permet entre autres de détecter les erreurs du genre "IF sans THEN", "DO sans LOOP", etc.

(;CODE) ---

Ce mot est un mot de très bas niveau (au moins en ce qui concerne l'utilisation). Son action est de mettre dans le cf du mot en cours de définition l'adresse qui suit (;CODE) lui-même. Le mot (;CODE) doit donc être suivi par du langage machine...
Il s'agit d'un mot système, utilisé par DOES > ...

(INTRP) ---

Mot système, appelé par l'interpréteur (voir INTERPRET).
C'est un des mots fondamentaux du langage, c'est lui qui organise les actions exécutées par votre ordinateur, il est en quelque sorte le chef d'orchestre...
Son action est la suivante :
Il lit le mot suivant du flot d'entrée, et le cherche dans le dictionnaire (dans le vocabulaire contexte).
S'il le trouve, il l'exécute ou le compile (suivant le mode actuellement positionné).
S'il ne le trouve pas, il essaie de le transformer en nombre (simple ou double longueur, et compte tenu de la base utilisée). En cas de succès, il le dépose sur la pile (si on est en mode "exécution") ou le compile comme littéral (si on est en mode "compilation"). En cas d'échec, le mot n'est ni un mot connu ni un nombre, et (INTRP) affiche un message "UK ERROR" (mot inconnu) et nettoie les piles.

, n ---

Alloue deux octets à la fin du dictionnaire (à l'adresse donnée par HERE), et y range le nombre n. Surtout utilisé pour définir des tables.

```
TABLE CARRES 1 , 4 , 9 , 16 , 25 , 36 , 49 ,
                ( Range dans la table CARRES les nombres 1, 4, 9, 16, 25, 36 et 49)
3 CARRES . 16 OK
```

Dans la littérature FORTH, on dit aussi parfois que le mot " " compile le nombre n à la fin du dictionnaire... Voir aussi le mot C,.

: ---

Mot de définition utilisé sous la forme :

: < nom > ... ;

Son action est complexe, et consiste d'abord à positionner le vocabulaire qui contiendra le mot à définir (place le contenu de CURRENT dans CONTEXT en prévision de la recherche ultérieure, voir ci-dessous), puis à créer une entrée dans le dictionnaire (plus précisément dans le vocabulaire CURRENT) pour le mot < nom > (fabrique le nf, le lf, le cf de < nom >), et entre en mode compilation.
Le processus de la compilation est le suivant (voir le fonctionnement de (INTERP) en mode compilation) :
Les mots qui constituent la définition sont isolés par l'interpréteur, et sont traités l'un après l'autre. Si un mot est immédiat, il est exécuté. Sinon, il est recherché dans les vocabulaires CONTEXT puis FORTH...
S'il est trouvé, son cfa est mis au dictionnaire à la suite des autres dans la définition (et ainsi de suite jusqu'à la rencontre du mot ";" qui marque la fin de la compilation)...
S'il n'est pas trouvé, le mot ":" essaie de le convertir en nombre, compte tenu de la base. S'il y parvient, il le compile en tant que "littéral", sinon on sort du mode compilation et un message d'erreur est édité (dans ce cas, la définition reste incomplète et il convient de l'oublier, en utilisant FORGET. Voir chapitre 8).

; ---

Termine une définition commencée par ":" et sort du mode compilation.
Toute définition commencée par ":" doit être terminée par ";". Si vous avez commencé une définition et que vous changez d'avis, le plus propre est de la terminer par un ";", puis de l'oublier par FORGET avant de continuer.
Si vous tapez une définition au clavier (dans ce cas, le flot d'entrée comporte un maximum de 8Ø caractères), il convient de la terminer avant le quatre-vingtième caractère, ou de taper la touche ENTREE (pour compiler partiellement le mot en cours de définition, et repartir à zéro dans le flot d'entrée sans quitter le mode compilation), sous peine d'erreur.

;S ---

Stoppe l'interprétation d'un écran de l'éditeur. Si vous voulez ne prendre en compte que les cinq premières lignes de votre écran (par exemple pour que le reste de l'écran soit considéré comme une remarque), il vous suffit d'insérer un ;S entre ces cinq premières lignes et la suite. ;S est aussi le mot d'exécution compilé par le mot ";" à la fin d'une définition par :...; et qui, lors de l'exécution, renvoie l'interpréteur interne à la suite de la procédure d'appel.

?COMP ---

Ce mot teste si on est bien en mode compilation, et renvoie le message d'erreur "CO ERROR" (Compilation Only) le cas échéant. Appelé par tous les mots qui sont conçus pour n'être utilisés qu'en mode compilation (instructions de branchement, pour l'essentiel). Si vous définissez un tel mot (à ne pas utiliser en mode exécution), vous pouvez y introduire le mot ?COMP. Votre mot testera alors automatiquement le mode, et bénéficiera des messages d'erreur du FORTH.

?CSP ---

Compare la valeur du pointeur de la pile des données avec la valeur contenue dans la variable CSP. Si ces deux valeurs sont différentes, émet le message d'erreur "UF ERROR" (Unfinished).
?CSP est appelé automatiquement par le mot ";". De sorte que si les mots immédiats que vous avez utilisés n'ont pas laissé la pile comme elle était en début de compilation, vous aurez ce message d'erreur. Par exemple, des mots comme IF, ELSE, THEN, BEGIN, CASE... utilisent la pile. IF empile des valeurs que THEN dépile. Un IF sans THEN sera donc détecté par ce procédé.

?EXEC ---

Teste si on est bien en mode "exécution", et affiche le cas échéant le message "XO ERROR" (eXecution Only). Vous pouvez introduire ?EXEC dans les mots dont vous ne voulez pas qu'ils soient utilisés en mode compilation.
?EXEC est à mettre en parallèle avec le mot ?COMP.

?S ---

Vérifie l'état de la pile, et affiche un message d'erreur si le pointeur de pile est hors des limites permises par la configuration du système (voir annexe relative à l'implantation en mémoire). Les messages affichés sont "ES" pour "Empty Stack" ou "FS" pour "Full Stack" (pile vide, ou pile pleine).
Cette vérification de base n'est faite qu'à la fin du traitement du flot d'entrée, de sorte que FORTH a peut-être déjà lu ou écrit en dehors des limites de la pile. Il appartient au programmeur de gérer correctement la pile des données !!!!... Pour la plupart des applications, 128 octets sont amplement suffisants (64 nombres sur la pile !).
Si toutefois vous utilisez intensivement la pile, par exemple lors d'applications utilisant la récursivité, il est à conseiller d'employer ?S avant d'empiler de nouvelles données. Le danger serait de continuer à écrire, par exemple dans la pile des retours, et alors tout peut arriver... Le plus probable est le plantage complet du système.

ALLOT n ---

Alloue n octets à la fin du dictionnaire, sans modifier le contenu de ces octets. Utilisé pour réserver des zones de stockage (voir les mots ARRAY, CARRAY, etc.)
On peut par exemple définir une variable sans l'initialiser par :

```
CREATE VAR     ( Pour fabriquer l'en-tête)
2 ALLOT        ( Réserve deux octets dans le pfa de VAR)
```

En écrivant 12 ALLOT, on réserverait de la place pour une table ou un tableau de six nombres, ou encore de douze caractères. Une utilisation courante est le cas où vous voulez définir une table, mais ne pas la remplir immédiatement. Par exemple, pour une table des carrés comportant 10 termes :

```
TABLE CARRES 20 ALLOT                          ( Réserve 20 octets)
: REMPLIR-TABLE ' CARRES                       ( donne le pfa de CARRES)
        11 1 DO I I *                          ( calcule le carré)
              OVER 1 - 2 * + !                 ( range dans la table)
            LOOP DROP ;
```

Puis : REMPLIR-TABLE FORGET REMPLIR-TABLE
Cette méthode évite que le mot REMPLIR-TABLE encombre le dictionnaire...
Noter que TABLE et CTABLE créent des mots ayant des pf vides, ainsi que CREATE.
ALLOT est aussi fréquemment utilisé lors de la définition de nouvelles structures de données (voir DOES > et CREATE).

ARRAY n ---

Un mot de définition utilisé sous la forme :

```
n ARRAY XXX
```

pour créer une entrée dictionnaire pour le mot XXX et réserver (voir ALLOT) n cellules 16 bits (2n octets) dans son pfa. Cela pour la partie "définition du mot XXX" (voir CREATE).
Lors de l'exécution du mot XXX (voir le mot DOES >), l'expression "p XXX" dépose sur la pile l'adresse du p-ième élément du tableau XXX. p n'est plus ici la dimension du tableau, c'est l'indice de l'élément auquel on veut accéder.

```
12 ARRAY VENTES/MOIS
1234 2 VENTES/MOIS !  ( 1234 ventes en février)
2 VENTES/MOIS ? 1234 OK
```

En fait, la définition de ARRAY est simplement

```
: ARRAY  ( n ---)
        CREATE 2* ALLOT                ( Réserve la place pour n nombres)
        DOES >  SWAP 2* + ;              ( Adresse du p-ième élément.
        adp = adØ + 2xp)
```

(Lors du début de l'exécution de p VENTES/MOIS, la pile contient p adØ --- , où adØ est le pfa de VENTES/MOIS, qui est aussi l'adresse du premier élément du tableau...
Voir DOES >).

ARRAY$ n1 n2 ---

Mot de définition utilisé sous la forme :

```
n1 n2 ARRAY$ < nom >
```

pour créer un tableau de n2 chaînes de caractères de longueur maximum n1.
Voir ARRAY$, au chapitre 5 (chaînes de caractères).

C, n ---

Range, dans l'octet suivant disponible du dictionnaire, l'octet de poids faible du nombre n. Le pointeur du dictionnaire est incrémenté de 1.
C, sert le plus souvent à créer des tableaux de caractères (voir aussi CARRAY ou CTABLE).

CARRAY n ---

Analogue à ARRAY, mais avec des caractères, ou des nombres 8 bits.
C'est un mot de définition, exécuté sous la forme :

```
n CARRAY XXX
```

pour créer une entrée dictionnaire pour le mot XXX, et réserver n octets dans son pf.
Une exécution ultérieure de "p XXX" déposera sur la pile l'adresse du p-ième octet du tableau XXX (utilisable par C@, par exemple).

COMPILE

Mot système. Ne peut être utilisé que dans une définition par :...; Lorsqu'une telle définition (par exemple : XXX ... COMPILE YY ...;) est exécutée, la valeur 16 bits (ici le cfa de YY) qui suit le mot COMPILE dans le pf de XXX est copiée (compilée) à la fin du dictionnaire (par exemple, COMPILE CASE copiera lors de l'exécution le cfa de CASE dans la définition en cours).
COMPILE est le plus souvent employé à l'intérieur de mots immédiats, comme IF ou DOES> (par exemple, IF compile le mot ØBRANCH). A ce titre, son utilisation concerne plutôt des programmeurs expérimentés...

CONSTANT n ---

Un mot de définition, utilisé sous la forme :

```
n CONSTANT XXX
```

pour créer une entrée dictionnaire pour le mot XXX, et compiler dans son pfa le nombre n. Lorsque XXX sera exécuté, il laissera n sur la pile.

```
365 CONSTANT JOURS-PAR-AN OK JOURS-PAR-AN . 365 OK
```

L'utilisation de constantes améliore beaucoup la lisibilité des applications, en même temps qu'elle accélère la vitesse d'exécution, tout en diminuant la place occupée en mémoire, à partir d'un certain seuil.
Exemple : 12 CONSTANT MOIS
Cette définition occupe onze octets dans le dictionnaire (cinq pour le nf, deux pour le lf, deux pour le cf, et deux pour le pf). Ensuite, chaque utilisation de MOIS dans les mots qui suivent n'occupera plus que deux octets, au lieu de quatre (pour le nombre 12, qui devrait être compilé comme littéral). Dès qu'on aura utilisé plus de six fois la constante MOIS, on aura regagné la place perdue par sa définition...

CONTEXT --- adr

Variable contenant le nfa (adresse du nom) du dernier nom du vocabulaire actuellement sélectionné comme "contexte", c'est-à-dire utilisé en premier pour la recherche lors de l'exécution (si le mot à chercher n'est pas trouvé dans le vocabulaire contexte, la recherche continue éventuellement dans le vocabulaire FORTH, qui est un voca-

bulaire particulier, en ce sens qu'il peut être considéré comme un "tronc commun" à tous les autres vocabulaires).
Voir le mot VOCABULARY.

```
CONTEXT @ @ ID.                    ( Affiche le nom du dernier mot défini)
EDITOR CONTEXT @ @ ID. EDSCR OK
```

EDSCR est le dernier mot du vocabulaire de l'éditeur.
Ne confondez pas le vocabulaire "contexte" avec le vocabulaire "courant" (voir CURRENT).

CREATE

Mot de définition, utilisé sous la forme : CREATE XXX
pour créer une entrée dictionnaire pour le mot XXX (fabrique son nf, son lf, et son cf), sans réserver de place pour son pf. Lorsque par la suite XXX sera exécuté, le pfa de XXX viendra sur la pile. CREATE sert surtout, en association avec DOES > (voir ce mot), pour définir d'autres mots de définition. Par exemple CONSTANT, VARIABLE, ARRAY, TABLE, STRING, CARRAY, CTABLE, ARRAY$,... sont des mots définis en utilisant CREATE et DOES > .

CSP --- adr

Pour "Current Stack Pointer"... Variable sauvegardant la valeur du pointeur de pile, et utilisée lors de la compilation (pour vérifier que cette dernière s'est bien passée).
CSP est typiquement un mot système, dont l'utilisation devrait être réservée aux programmeurs expérimentés qui désirent par exemple réécrire leur propre interpréteur FORTH...

CTABLE ---

Mot de définition, exécuté sous la forme : CTABLE XXX
pour créer une entrée dictionnaire pour le mot XXX, sans réserver de place dans son pf. La table d'octets XXX ainsi créée sera remplie ensuite, le plus souvent en utilisant le mot C,. Lorsqu'ensuite on exécutera "n XXX", la valeur du n-ième octet de la table apparaîtra sur la pile (elle sera mise sur la pile en tant que nombre 16 bits, avec l'octet de poids fort à Ø).
Attention, le remplissage de la table par C, doit se faire avant toute

modification du dictionnaire... Il est possible de la remplir à plus tard, mais il faudra dans ce cas réserver de la place dans le pf de XXX, par le mot ALLOT...
Par exemple : CTABLE BITS
Méthode 1 :

```
1 C, 2 C, 4 C, 8 C, 16 C, 32 C, 64 C, 128 C,
```

Méthode 2 :

```
8 ALLOT                          ( Réserve de la place pour huit octets)
```

Puis:

```
: REMPLIR-LA-TABLE 1
  8 0 DO DUP ' BITS I + C! 2* LOOP DROP ;
REMPLIR-LA-TABLE  FORGET REMPLIR-LA-TABLE
```

Ces deux méthodes donnent le même résultat. La première est plus simple, mais doit être utilisée avant toute nouvelle définition dans le dictionnaire. La deuxième permet de définir d'autres mots avant de revenir donner des valeurs aux éléments de la table.

```
: VOIR-BITS 8 0 DO I BITS . LOOP ;
VOIR-BITS  1 2 4 8 16 32 64 128 OK
```

CURRENT --- adr

Variable donnant le nfa du dernier mot du vocabulaire "courant", celui dans lequel seront compilées les prochaines définitions.
La variable CURRENT peut être positionnée par le mot DEFINITIONS. Voir VOCABULARY.

DCONSTANT d ---

Analogue à CONSTANT mais opère sur des nombres 32 bits. (définit une constante 32 bits).
Exemple : lecture d'un paramètre au début d'un traitement...

```
." Salaire maximum : "  INPUT DCONSTANT SAL-MAX
                        ( Le nombre entré sera mis dans la constante SAL-MAX)
```

DEFINITIONS ---

Sélectionne le vocabulaire "contexte" comme étant aussi le vocabulaire "courant". Les prochaines définitions seront donc placées dans le vocabulaire précédemment sélectionné comme contexte.
Voir VOCABULARY.

DLITERAL d ---

Même chose que LITERAL mais traite un nombre 32 bits. Voir LITERAL.

DOES >

Marque le début de l'action que devra réaliser (en mode exécution, lorsqu'il sera appelé) un mot créé par un mot de définition écrit en FORTH. Plus précisément :

```
: XXX < actions1 > CREATE < actions2 > DOES > < actions3 > ;
                                    ( XXX est le nom d'un mot de définition)
... suivi de XXX YYY                ( Crée le mot YYY dont le "type" est XXX)
```

Etudions d'un peu plus près le processus de cette création .
Lorsqu'on tape XXX YYY, tout d'abord, < actions1 > est exécuté (c'est le début de XXX).
Puis, CREATE crée une entrée dictionnaire pour le mot YYY (avec les vérifications d'usage...).
Enfin, < actions2 > s'exécute (consiste en général à compléter la définition du mot YYY).
Lorsqu'on exécute le mot YYY, c'est < actions3 > qui s'exécute, avec le pfa de YYY sur la pile (le mot < actions3 > utilise comme paramètres ce que < actions2 > a mis dans le pf de YYY). On dit parfois que YYY est un "fils" de XXX...
Un exemple simple et concret (et classique): le mot CONSTANT

```
-1-  :  CONSTANT  CREATE  ,  DOES >  @ ;
     CONSTANT NU OK
-2-  365 CONSTANT JOURS OK
-3-  JOURS . 365 OK
```

Le déroulement des opérations est le suivant :
-1- Définition du mot CONSTANT (le message "CONSTANT NU" nous rappelle que le mot existe déjà...).

-2- On fabrique un nouveau mot, JOURS. C'est la première partie de CONSTANT qui s'exécute : FORTH fabrique l'en-tête du mot JOURS, et met le nombre 365 dans son pf (par l'intermédiaire du mot "," qui suit CREATE).
-3- JOURS est maintenant un mot du dictionnaire, on l'exécute. C'est la deuxième partie de CONSTANT (c'est-à-dire le @ qui suit DOES >) qui entre en action : l'adresse de JOURS étant sur la pile (il s'agit plus exactement de l'adresse du pf de JOURS, où est actuellement rangé le nombre 365), le mot @ va chercher le contenu de cette adresse, et le dépose sur la pile ce qui est bien le genre d'action qu'on peut attendre d'un mot qui a été déclaré du type "constante".

Autre exemple :

```
: DARRAY( n --- Création de tableaux de nombres 32 bits )
        CREATE 2* 2* ALLOT      ( Réserve la place pour n nombres 32 bits)
        DOES > SWAP 2* 2*  ( Dépose sur la pile l'adresse de début du)
        + ;     ( tableau, augmentée de 4p, où p= N° de l'élément considéré)
```

On pourra alors utiliser :

```
25 DARRAY BINOMBRES          ( Pour créer un tableau de 25 nombres 32 bits)
INPUT 13 BINOMBRES D!                ( Pour affecter l'élément numéro 13)
13 BINOMBRES D@ D.                ( Pour afficher la valeur de l'élément 13)
```

Les tableaux ainsi créés ne sont pas initialisés (à la création, les valeurs sont indéterminées). On aurait pu utiliser par exemple dans la partie "CREATE" l'expression :

```
CREATE 0 DO 0 , 0 , LOOP DOES > ...
```

pour créer le tableau en initialisant toutes les valeurs à 0 ...
Ou encore, si vous voulez des tableaux dont le premier élément soit numéroté 1 au lieu de 0, remplacer la partie "DOES > " par :

```
...DOES > SWAP 1- 2* 2* + ;
```

DVARIABLE ---

Mot de définition utilisé sous la forme :

```
DVARIABLE XXX
```

pour créer un mot XXX (dont le pf est une zone de quatre octets), et qui, lorsqu'on l'exécute, laisse sur la pile l'adresse de son pf. A la création, ce pf est initialisé avec la valeur 32 bits Ø.
Pour lire le contenu d'une variable, il suffit d'utiliser :

```
XXX D@                    ( Dépose sur la pile le contenu du pf de XXX)
```

Pour y ranger une nombre 32 bits d, on utilise :

```
d XXX D!                  ( Dépose dans le pf de XXX le nombre d)
```

EDITOR ---

Sélectionne le vocabulaire de l'éditeur comme contexte.

```
' RAZ UK ERROR OK
      ( RAZ est une commande de l'éditeur, remettant le block en cours à blanc)
EDITOR
' RAZ . 15235 OK     ( La recherche s'effectue dans le vocabulaire de l'éditeur)
' DUP . 721 OK           ( La recherche continue dans le vocabulaire FORTH)
```

EXECUTE adr ---

Exécute le mot FORTH dont le cfa est sur la pile. Ce mot est surtout utilisé par le système, mais il peut être commode pour appeler un mot de manière indirecte (voir l'annexe D),ou dans une application utilisant un grand nombre de commandes. Dans cette dernière situation, on peut remplacer CASE ou GODO par l'emploi d'une table de commandes, contenant les cfa des mots à exécuter. Voir l'exemple du mot TABLE.

FORTH ---

Sélectionne le vocabulaire FORTH comme contexte. En fait, ce mot sert surtout à déselectionner un vocabulaire antérieurement positionné comme contexte.

```
EDITOR VLIST .....
FORTH VLIST .....
```

Les mots de l'éditeur ont disparu.

IMMEDIATE ---

Sert à marquer comme immédiat le dernier mot compilé dans le dictionnaire. Ce dernier mot sera donc par la suite exécuté dès qu'il sera appelé, même pendant la compilation d'un autre mot.
IMMEDIATE n'est pas lui-même un mot immédiat, de sorte qu'il convient de l'utiliser en dehors d'une définition par :...;. Sinon, il rendrait immédiat le dernier mot du dictionnaire lors de l'exécution de l'application (ou non immédiat, si ce mot était déjà immédiat...).

```
: GAG CR 10 SPACES ." Je suis un gag immédiat "  BELL ; IMMEDIATE
: XX ." Voici la liste des mots du FORTH : "  GAG VLIST ; ( Touche ENTREE)
  Je suis un gag immédiat OK
```

Le mot GAG a été exécuté pendant la compilation de XX, et n'a pas été compilé dans la définition de XX. (Si maintenant vous tapez XX, vous obtiendrez seulement la phrase "Voici la liste des mots du FORTH", suivie de la liste du dictionnaire).
Au point de vue "système", FORTH reconnait les mots immédiats en testant le septième bit (le bit 6) de l'octet de longueur du nf (voir l'annexe C pour une description des bits de cet octet). Si ce bit est à 1, le mot est immédiat. Sinon, il ne l'est pas.

```
2 BASE ! ' IF NFA C@ .11000010 OK     ( IF est immédiat)
         ' OR NFA C@ . 10000010 OK    ( OR ne l'est pas...)
```

INTERPRET ---

INTERPRET exécute le mot dont le cfa est contenu en position 4 dans le tableau SYSVEC. De base, ce mot est (INTRP) (interpréteur par défaut). Il vous est possible d'écrire un autre mot, par exemple ((INTRP)), et de placer son adresse dans SYSVEC. Eh oui, en FORTH on peut même modifier l'interpréteur... par exemple pour utiliser un autre type de nombres, flottant ou 64 bits...
INTERPRET est le cœur de l'interpréteur externe. Voir l'annexe C pour une description sommaire du fonctionnement de FORTH.

LIT --- n

LIT est un mot système, qui est compilé (en général par le mot LITERAL, que ce soit par le système, ou de manière volontaire, par l'utilisateur) dans une définition pour y placer un nombre 16 bits en tant que littéral.

```
: ORIGINE 50 150 ;
```

Les nombres 50 et 150 sont compilés comme littéraux, c'est-à-dire qu'ils font partie de la définition du mot ORIGINE, et apparaîtront sur la pile lors de l'exécution du mot (''littéralement'', comme si on venait de les taper en mode direct). C'est exactement ce que fait le mot LIT (sa raison d'être est d'éviter que l'interpréteur interne du FORTH ne considère les nombres 50 et 150 comme des cfa et ne les exécute, ce qui n'aurait aucun sens... Voir annexe C).
Lors de la compilation, le pf de ORIGINE se construit de la manière suivante :

```
cfa de LIT nombre 50 cfa de LIT nombre 150 ...
```

Lors de l'exécution de ORIGINE, LIT est appelé deux fois. A chaque fois, il lit le nombre qui le suit dans la définition (la première fois 50, la deuxième fois 150), et le place sur la pile.

LITERAL n ---

LITERAL est le mot effectivement utilisé pour compiler le nombre n en tant que littéral dans la définition en cours.
LITERAL n'a pas d'action en mode exécution. Son action en mode compilation est de mettre, dans le pf du mot en cours de définition, le cfa de LIT, suivi du nombre n (qui est actuellement sur la pile).
LITERAL est assez rarement employé par l'utilisateur (mais souvent par l'interpréteur). On l'emploie surtout lorsqu'on est provisoirement sorti du mode compilation pour faire un calcul (et qu'on veut compiler comme littéral le résultat de ce calcul).

```
: XX ...                ( On veut à ce point de la définition compiler 234 x 23)
[ 234 23 * ] LITERAL ... ;
```

Voir les mots [et] pour quitter ou retrouver le mode compilation.
Dans l'exemple, c'est le résultat de la multiplication qui sera compilé dans le mot, exactement comme si on avait fait le calcul à part, et tapé simplement le nombre 5382 (sans le mot LITERAL) dans la définition.

On aurait le même résultat lors de l'exécution, en tapant simplement "234 23 "*, à cela près que la multiplication s'effectuerait à chaque exécution de XX.
Le pf de XX contient :
Si on a utilisé "234 23 *" :

```
... cfa de LIT nombre 234 cfa de LIT nombre 23 cfa de * ...
```

Si on a utilisé "[234 23 *] LITERAL" :

```
... cfa de LIT nombre 5382 ...
```

Il est bien clair que cette dernière version est plus rapide à l'exécution, tout en prenant moins de place dans le dictionnaire.

SMUDGE ---

SMUDGE est un mot immédiat qui positionne ou dépositionne le "bit de smudge" du dernier mot créé au dictionnaire. Le bit de smudge est le bit 5 (le sixième bit) de l'octet de longueur du nf de la définition considérée.
Le système l'utilise pour savoir si la définition est terminée, et pour interdire la récursion (en effet, FORTH ne disposant pas de variables locales, il n'est pas possible d'y faire de la récursion "à l'aveugle", comme on la conçoit le plus souvent. La récursion est possible, mais c'est à l'utilisateur de gérer lui-même la pile et les sauvegardes des variables).
De sorte que l'une des actions du mot ":" (début de définition) est de mettre à un le bit de smudge pour la définition en cours. A partir de là, ce mot est introuvable au dictionnaire, et donc ne peut s'appeler lui-même. En fin de définition, le mot ";" rétablit la situation normale (bit de smudge à zéro).
Il est possible de "tromper" le système en dépositionnant artificiellement ce bit en cours de définition. Il ne faudra pas oublier de le rétablir avant le mot ";", pour que ce dernier mot le remette effectivement à zéro (sinon le mot serait introuvable au dictionnaire).
Exemple : fonction factorielle
On note d'habitude :

$$n! = n(n-1)(n-2)... \times 2 \times 1$$

la factorielle du nombre n, avec la convention que la factorielle du nombre Ø vaut 1.
Autrement dit : si $n = Ø$, alors $n! = 1$, sinon $n! = n \times ((n-1)!)$

D'où le mot :

```
: FACT    ( un --- ud )
      DUP Ø=
      IF DROP 1.    ( Factorielle de Ø vaut 1)
      ELSE DUP 1-
          SMUDGE  ( Dépositionne le bit de smudge)
          FACT    ( Factorielle de n-1)
          SMUDGE  ( Repositionne le bit de smudge)
                  ( En prévision du point-virgule)
          ROT UDN*( Multipliée par n)
      THEN ;
```

Puis: 5 FACT UD. 12Ø OK

Si vous omettez les mots SMUDGE dans la définition ci-dessus, la compilation échouera, et vous aurez un message "FACT ?,UK ERROR". Simplement parce que le FACT qui est entre les deux SMUDGE n'aura pas été trouvé au dictionnaire, puisqu'à ce moment de la compilation le bit de smudge du mot FACT (en-tête de la définition en cours de compilation) est positionné à 1.

Par ailleurs, cette possibilité de récursivité est bien entendu limitée par la taille de la pile des données. Il est toutefois prévu que vous puissiez reconfigurer l'implantation mémoire du système, et en particulier augmenter la taille des piles. Voir à ce sujet l'annexe E, relative à l'organisation mémoire du FORTH.

STATE --- adr

Variable utilisée par le système pour savoir si on est en mode compilation, ou en mode exécution. La valeur Ø indique le mode exécution. La valeur de la variable STATE est testée par l'interpréteur, pour savoir en quoi consiste le traitement du flot d'entrée (compiler ou exécuter).

STRING n ---

Mot de définition utilisé sous la forme "n STRING < nom > " pour créer une variable chaîne de caractères, pouvant contenir des chaînes de longueur maximum n caractères. Voir STRING, au chapitre 5 (chaînes de caractères).

TABLE ---

Mot de définition, exécuté sous la forme :

```
TABLE XXX
```

pour créer une entrée dictionnaire pour le mot XXX, sans réserver de place dans son pf. La table XXX ainsi créée pourra contenir des nombres 16 bits. Elle sera remplie ensuite, le plus souvent en utilisant le mot "," (ou par une méthode analogue à celle qui est décrite dans le mot CTABLE).
Lorsqu'ensuite on exécutera "n XXX", la valeur du n-ième nombre 16 bits de la table apparaîtra sur la pile.
Une utilisation intéressante des tables est la notion de "table de commandes". Le contenu de la table est alors une suite d'adresses, qui sont des cfa de mots FORTH. L'éditeur est construit sur ce principe.
Exemple schématique :

```
: MOTØ ... ;    : MOT1 ... ; ... :    MOT9 ... ;
```

Puis: TABLE COMMANDE FIND MOTØ , FIND MOT1 , ... FIND MOT9 ,
On pourra ensuite utiliser cette table par exemple comme ceci :

```
: MENU BEGIN KEY 48 MAX 57 MIN          ( Codes ASCII de "Ø" à "9")
            48 - COMMANDE EXECUTE AGAIN ;
```

Si vous tapez Ø (code 48), c'est MOTØ qui sera exécuté.
Si vous tapez 1 (code 49), c'est MOT1 qui sera exécuté.
Et ainsi de suite, le code de la touche tapée servant d'indice dans la table.
A chaque fois, la séquence "KEY COMMANDE" dépose sur la pile l'adresse de la commande à exécuter. Le mot EXECUTE se charge de l'appel du mot correspondant.
Lorsqu'il y a un choix entre beaucoup d'options, cette méthode est plus simple à mettre en oeuvre que CASE ou GODO.

VARIABLE ---

Mot de définition utilisé sous la forme VARIABLE XXX pour créer un mot XXX (dont le pf est destiné à contenir un nombre 16 bits), et qui, lorsqu'on l'exécute, laisse sur la pile son pfa. Lors de la création, le pf de XXX est initialisé avec la valeur 16 bits.
Pour en lire le contenu, il suffit donc d'utiliser :

```
XXX @      ( Dépose sur la pile le contenu du pf de XXX)
```

Pour y ranger une nombre 16 bits n, on utilise :

```
n XXX !    ( Dépose dans le pf de XXX le nombre n)
```

VOC-LINK --- adr

Variable système pointant sur la deuxième cellule du pfa du dernier vocabulaire créé. Cette cellule contient à son tour l'adresse de la deuxième cellule du pfa du vocabulaire précédent si ce vocabulaire n'est pas FORTH, et Ø pour FORTH. VOC-LINK réalise ainsi un chaînage entre les différents vocabulaires. Voir l'annexe C pour plus de précisions.

VOCABULARY ---

Mot de définition utilisé sous la forme "VOCABULARY XX" pour créer un nouveau vocabulaire. FORTH emploie des vocabulaires pour ne pas surcharger le dictionnaire, et pour accélérer le processus de la compilation. Chaque vocabulaire est en quelque sorte un sous-dictionnaire séparé, qui viendrait s'ajouter au FORTH.
FORTH lui-même constitue un vocabulaire, l'éditeur (EDITOR) en est un autre...
Les vocabulaires peuvent être activés pour la recherche-exécution de mots (vocabulaire contexte), ou pour la définition de nouveau mots (vocabulaire courant).
Votre FORTH autorise la présence de huit vocabulaires simultanément dans le dictionnaire, mais il n'est possible d'utiliser qu'un seul vocabulaire autre que FORTH à un moment donné.
Dans la pratique : créons un vocabulaire pour dessiner en perspective...
Pour créer le vocabulaire, par exemple PERSPEC :

```
VOCABULARY PERSPEC
```

A l'exécution, PERSPEC sélectionnera le vocabulaire PERSPEC comme contexte.
Pour y définir de nouveaux mots :

```
PERSPEC DEFINITIONS     : FUITE ... ;
                        : FACE ... ; etc.
VLIST ...Nos mots sont bien là...
FORTH VLIST ...Ils ont disparu, et ne peuvent être exécutés
PERSPEC VLIST ...Ouf, les revoici...
FORTH VLIST ...Disparus de nouveau, on recommence
: MOT RED FRAME ; VLIST ...Les revoici.
```

Nous n'avions pas tapé "FORTH DEFINITIONS", de sorte que MOT a été défini dans le vocabulaire PERSPEC, repositionnant auto-

matiquement le vocabulaire courant (PERSPEC) comme contexte. FORTH DEFINITIONS sélectionne maintenant FORTH comme contexte et courant, de sorte que les prochaines définitions se feront dans le vocabulaire FORTH.

```
MOT UK ERROR OK
```

MOT est inconnu du vocabulaire FORTH...
Par contre, nous pouvons maintenant définir :

```
: MOT BLUE FRAME ;
```

sans recevoir le message "not unique".
Il est possible d'utiliser des noms différents dans des vocabulaires différents, sauf dans FORTH...
PERSPEC MOT donne un pourtour d'écran rouge, FORTH MOT redonne du bleu.
Mais: PERSPEC DEFINITIONS FORGET MOT MOT
redonne du bleu, et une nouvelle définition de MOT par :

```
: MOT YELLOW FRAME ;
```

provoquera un message "Not unique", car le mot MOT figure déjà dans le vocabulaire FORTH (PERSPEC n'est jamais qu'un prolongement de FORTH).
Pour les applications, il sera souvent commode de définir les vocabulaires comme immédiats (voir IMMEDIATE).

```
: TOUR FRAME BELL 100 WAIT ;
VOCABULARY JEU IMMEDIATE
JEU DEFINITIONS     : COULEUR RED;       : ZOU COULEUR TOUR ;
FORTH DEFINITIONS : COULEUR YELLOW ; : ZOU COULEUR TOUR ;
: JEJOUE BEGIN JEU ZOU FORTH ZOU PTRIG UNTIL ;
```

Si vous n'avez pas positionné JEU comme immédiat, JEJOUE devra s'écrire :

```
: JEJOUE BEGIN [ JEU ] ZOU
                [ FORTH ] ZOU PTRIG UNTIL ;
```

Et bien sûr, dans cet exemple, pas un seul message "Not Unique"...
Pour d'autres détails sur l'organisation mémoire des vocabulaires, voir l'annexe C.

[---

Mot immédiat qui permet de sortir du mode compilation et positionne le mode exécution. Ce mot est le plus souvent utilisé en liaison avec] pour interrompre momentanément une compilation, par exemple pour un calcul auxiliaire (voir aussi LITERAL ou DLITERAL), pour changer de base, ou pour vérifier l'état d'une mémoire, faire un VLIST, etc.
Le caractère [s'obtient au clavier en tapant CNT-A.

```
: MASQUE    ( on est en décimal)
 157 [ 2 BASE ! ] Ø1111111 AND ;
```

Lors de la compilation, 157 sera interprété en décimal, et Ø1111111 en base 2.
Vérification : DECIMAL MASQUE U. 29 OK
(En effet, 157 s'écrit 1ØØ111Ø1 en base deux, donc le résultat est le nombre qui s'écrit 111Ø1 en base deux, c'est-à-dire 29.)
Cet exemple est sans intérêt, mais montre bien l'utilisation de la sortie provisoire du mode compilation pour agir sur l'interprétation du flot d'entrée qui suit (ici en changeant de base).

[COMPILE] ---

Mot immédiat, utilisé lors d'une compilation pour forcer la compilation du mot qui suit dans le flot d'entrée. L'intérêt de ce mot est de pouvoir compiler des mots immédiats (qui autrement seraient exécutés, même en mode compilation).
Vous pouvez vous faire facilement, si vous le désirez, une version française du FORTH... Il suffit de redéfinir les mots...
Par exemple :

```
: DRELIN BELL ; : COULEUR COLOR ; : DUREE DURATION ; etc.
```

Le problème sera pour les mots immédiats. La solution est simple, il faut forcer leur compilation, puis les rendre immédiats...
Ex :

```
: REPETER [COMPILE] BEGIN ; IMMEDIATE
: FIN-REPETER [COMPILE] UNTIL ; IMMEDIATE
: SI ;          : ALORS [COMPILE] IF ; IMMEDIATE
: SINON [COMPILE] ELSE ; IMMEDIATE
: FINSI [COMPILE] THEN ; IMMEDIATE
```

Vous pourrez alors même utiliser une structure du genre :

```
SI < condition >
   ALORS < actions1 >
   SINON < actions2 >
FINSI      ( Le paradis de la programmation structurée !!!)
```

Exemple :

Le mot ' (apostrophe) est un mot immédiat, qui laisse sur la pile le cfa du prochain mot du flot d'entrée. En mode compilation, le flot d'entrée est constitué par la suite de la définition.
Si vous voulez utiliser le mot ' à l'intérieur d'une définition, il vous faut donc en forcer la compilation. De cette manière, il n'entrera en action qu'au moment de l'exécution de la définition, à un moment où le flot d'entrée ne sera plus la définition elle-même.
Par exemple, pour faire un "dump" d'un mot FORTH...

```
: DMP ' NFA 1 DUMP ;
```

réalisera le dump du mot NFA, dans tous les cas. Cela est bien sur d'une utilité toute relative !!!
La bonne méthode est :

```
: DMP [COMPILE] ' NFA 100 DUMP ;
```

Vous pourrez alors taper DMP VLIST pour obtenir un dump du mot VLIST (100 octets seulement), ou encore DMP NFA pour obtenir un dump du mot NFA. Bref, le mot DMP peut nous donner un dump de n'importe quel mot FORTH...
Cette méthode est souvent utilisée pour faire des mots "vectorisés", pouvant accomplir des actions avec d'autres mots comme paramètres (voir l'annexe D).
Exemple : un mot COURBE, qui pourra tracer une courbe relative à une fonction que l'utilisateur pourra choisir au moment de l'exécution. Il suffit par exemple de réserver une variable 'FN pour ranger le cfa de la fonction utilisée, et de commencer la définition de COURBE par :

```
: COURBE [COMPILE] ' 'FN ! ... ;
```

En cours de définition, pour calculer y en fonction de x, il suffira de faire "'FN EXECUTE", au lieu du calcul dans un cas particulier.
Si ensuite vous définissez par exemple :

```
: Y=X2     ( x --- y ) DUP * ;
: Y=X3/3   ( x --- y ) DUP Y=X2 3 */ ;
                              ( ou d'autres fonctions plus compliquées,...)
```

Vous pourrez alors utiliser simplement :
COURBE Y=X2 ou COURBE Y=X3/3 etc. pour tracer la courbe considérée.

Notez que dans cet exemple, les fonctions peuvent très bien être définies postérieurement au mot COURBE, ainsi d'ailleurs que les mots à dumper dans l'exemple précedent. Ceci donne à cette technique une très grande puissance.

] ---

Mot immédiat. Réalise l'opération inverse de celle du mot], c'est-à-dire repositionne le mode compilation. Le mot] n'est pratiquement utilisé qu'en liaison avec [(voir ce mot), pour continuer une compilation provisoirement interrompue.
Le caractère] s'obtient au clavier en tapant CNT-Z.

Chapitre 5

Chaînes de caractères Conversions entre données numériques et alphanumériques

Chaînes de caractères

Les chaînes de caractères sont caractérisées par trois paramètres : leur longueur maximum (au plus 255 caractères), leur longueur courante, et l'adresse à partir de laquelle sont rangés les caractères qui les composent. Du point de vue de l'utilisateur, les mots qui les gèrent prennent en général comme paramètres sur la pile, l'adresse de rangement et la longueur courante.
Dans ce qui suit, nous écrirons le plus souvent :

< str --- > au lieu de < adr n --- >

pour décrire les paramètres de la pile.
De la même manière, nous noterons (str) la chaîne de caractères dont les paramètres (adresse et longueur) sont représentés par str.

Utilisation des chaînes de caractères :
Il y a deux facons de définir des chaînes. On peut définir des *variables chaînes* (voir STRING), qui nécessitent des affectations, mais ne posent aucun problème d'utilisation. D'autre part, on peut utiliser des chaînes de manière "directe", en utilisant le mot "guillemet".

Ces chaînes sont alors rangées dans une zone temporaire, où elles peuvent être effacées par le système, sans avertissement. On parle alors de *chaînes temporaires*. Si on veut les réutiliser, il est donc conseillé de les mettre le plus tôt possible dans des variables, où elles seront plus en sécurité. Voir dans $ = un exemple du type de problèmes qui peuvent se présenter...
Une chaîne temporaire n'a pas d'attribut de longueur maximum (contrairement aux variables chaînes).
Les mots se référant à la longueur maximum ne sont utilisables que par les variables chaînes...

Conversion et formatage des entrées-sorties numériques

Lorsque vous entrez un nombre au clavier, il s'agit d'une chaîne de caractères que FORTH doit, pour l'utiliser dans les calculs, convertir en nombre.
De la même manière, les résultats des calculs qu'a faits votre ordinateur doivent être reconvertis en chaînes de caractères pour que vous puissiez les lire.
Sinon, le nombre 67 (dans le système décimal), par exemple, serait affiché par EMIT en tant que caractère, et FORTH vous afficherait un "C". Pour afficher effectivement 67, il doit commencer par transformer le nombre en la chaîne composée des deux caractères 6 et 7 (codes ASCII 54 et 55).
En fait, FORTH fait mieux que cela, puisqu'il peut lire ou afficher des nombres dans n'importe quelle base, pas seulement en base dix.

Ce travail de conversion est en général réalisé par des mots système, et est le plus souvent transparent pour l'utilisateur. Toutefois, si vous avez besoin d'un format spécial pour vos nombres, FORTH met à votre disposition des mots de conversion de nombres double longueur en chaînes de caractères. Il s'agit des "mots en #", et des mots HOLD et SIGN.

La zone utilisée pour toutes ces conversions est la zone située entre la fin du dictionnaire et le début de la zone réservée au rangement temporaire des chaînes de caractères (voir le mot PAD).

" --- str

Mot immédiat, utilisé sous la forme " cccc" pour définir une chaîne de caractères.
Le premier guillemet est un mot FORTH, et à ce titre doit être suivi d'un blanc. Par contre, le deuxième est un terminateur ne faisant pas partie de la chaîne. S'il est précédé d'un blanc, ce blanc fera partie de la chaîne.
Le mot "guillemet" agit différemment en mode interactif ou en mode compilation.

En mode interactif :
crée une chaîne *temporaire* cccc, laissant sur la pile l'adresse et la longueur de la chaîne. Cette chaîne doit nécessairement comporter au moins un caractère. Elle est placée dans une zone temporaire que FORTH utilise de temps en temps, il est donc conseillé de l'utiliser au plus tôt...
Exemples :

```
" Les sanglots longs des violons de l'automne" TYPE
20 STRING NOM " Verlaine" NOM $! NOM TYPE
```

En mode compilation :
Compile dans la définition en cours le mot (") (voir ce mot), suivi des caractères constituant la chaîne. Lors de l'exécution de la définition, les paramètres de la chaîne seront déposes sur la pile.

"" --- str

Chaîne "vide", de longueur 0 (le seul moyen simple de l'obtenir, puisqu'une chaîne définie par le mot "guillemet" comporte toujours au moins un caractère).

```
10 STRING AA "" AA $! AA TYPE OK     A LEN . 0 OK
```

ud1 --- ud2

S'utilise lors d'une conversion avec formatage d'un nombre entier [double longueur (32 bits) et non signé] en chaîne de caractères.
ud2 est le quotient de ud1 par la base courante (voir BASE). Le reste de la division, qui représente le chiffre à afficher, est transformé vers le code ASCII du caractère à afficher, et mis au début de la chaîne

de caractères représentant le nombre en cours de conversion.
La transformation en chaîne de caractères s'effectue à partir du chiffre des unités (à partir de la droite...), un chiffre à la fois.
Le déroulement des opérations est le suivant :

<#	initialise le processus.
#	permet d'obtenir le chiffre suivant. C'est bien sûr le code ASCII du chiffre à générer qui est calculé...
#S	convertit le reste du nombre.
#>	termine le processus.

> laisse sur la pile l'adresse et la longueur de la chaîne de caractères qui représente l'écriture de ud1 dans la base courante.

Exemple : visualiser tous les bits d'un nombre 16 bits...

```
: 16-BITS ( n ---)
        BASE @ SWAP 2 BASE !   ( Sauvegarde la base en cours)
        S->D <# 16 0 DO # LOOP #> TYPE:.
        BASE ! ;               ( Restaure la base précédente)
10638 16-BITS 0010100011000111 0 OK
-10638 16-BITS 1101011001110010 OK
HEX 3A5F 16-BITS 0011101001011111 OK
```

S- > D transforme n en nombre 32 bits (voir chapitre 2).
16-BITS est plus compliqué, mais mieux que U, car les 16 bits sont affichés, même les zéros en début de représentation.
Lors de cette conversion, on peut aussi utiliser HOLD pour introduire des caractères (virgule, symboles divers) ou SIGN pour mettre un signe, si on le désire (le processus précedent ne traite que des nombres non signés), dans la chaîne de caractères résultat. Voir SIGN.

Autre exemple : (avec HOLD)

```
: FRIC  ( d --- str )
   DABS                  ( nombre entier 32 bits non signé)
   <#                    ( initialise la conversion)
   70 HOLD               ( écrit la lettre "F")
   32 HOLD               ( écrit un espace)
   # #                   ( pour obtenir les centimes)
   44 HOLD               ( pour écrire une virgule)
   #S                    ( pour obtenir les francs)
   #>  ;                 ( achève la conversion)
```

Puis : CR 345678. FRIC TYPE donnera 3456,78 F OK
Ne pas oublier le point à la fin du nombre pour indiquer un nombre entier 32 bits.
Ou encore (cf. *Starting Forth,* de Leo Brodie) :
conversion en heures, minutes, secondes...

```
: :00 # 6 BASE ! # DECIMAL 58 HOLD ;
```

(Une manière élégante de diviser par 60 : le premier # divise par 10, le deuxième divise par 6 avant de revenir en mode décimal et de faire précéder le reste de la division par 60 de deux points (ASCII 58)).

```
: HMS < # :00 :00 #S # > TYPE ;     ( Sans commentaire)
3600. HMS 1:00:00 OK
12345678. HMS 3429:21:18 OK     ( 3429 heures, 21 mn, 18s)
```

> d --- adr n

Termine le processus de conversion-formatage d'un nombre entier 32 bits en chaîne de caractères.
Remplace le nombre d (laissé par le processus de conversion) par l'adresse et la longueur de la chaîne résultat. Voir #, < #, #S.

#S ud --- 0 0

Convertit tout ud comme le feraient des # successifs, et laisse sur la pile le nombre 0 (nombre entier 32 bits). Voir #.
Ce 0 0 est nécessaire au bon fonctionnement du mot # >.

$! str1 str2 ---

Range la chaîne (str1) dans la variable chaîne (str2). [Remplace (str2) par (str1)]

```
10 STRING A     20 STRING B
" abcd" A $!    A TYPE abcd OK
A B $!          B TYPE abcd OK
```

$+ str1 str2 ---

Concatène la chaîne (str1) à la fin de la chaîne (str2), compte tenu de la longueur maximum de (str2). La chaîne (str2) doit être une variable chaîne (pour avoir une longueur maximum), alors que (str1) peut être une chaîne temporaire ou une variable chaîne.

```
10 STRING AA 12 STRING BB
" qw" AA $!   " erty" AA $+                    AA TYPE qwerty OK
" az" BB $!   " erty" AA $!        AA BB $+    BB TYPE azerty OK
" 123456789" BB $+ BB TYPE azerty1234 OK       ( longueur maximum 10)
```

$< str1 str2 --- cond

cond vaut 1 si (str1) est avant (str2) dans l'ordre lexicographique (comme dans un dictionnaire, à cela près que tous les codes ASCII sont utilisés, et pas seulement les lettres de l'alphabet). L'une au moins des deux chaînes doit être une variable chaîne (voir $= pour explication).

```
10 STRING XX " azerty" XX $! XX " qwerty" $< . 1 OK
XX " azerty " $< . OK                ( espace à la fin de la deuxième chaîne)
```

$= str1 str2 --- cond

cond vaut 1 si (str1) est égal à (str2) (même longueur et mêmes caractères). L'une au moins des deux chaînes doit être une variable chaîne.

```
10 STRING XX " azerty" XX $!    " qwerty" XX $= . 0 OK
" azerty " XX $= . 1 OK
" azerty " XX $= . 0 OK    ( longueurs différentes)
```

Piège classique :

```
" azerty" " qwerty" $= . 1 OK
```

Explication : les deux chaînes sont des chaînes temporaires, rangées au même endroit. « qwerty » a occupé la même place en mémoire que « azerty », de sorte que $= compare « qwerty » avec « qwerty », puisque les paramètres des deux chaînes sont identiques (même longueur).

Par contre :

```
" qwerty " " azertyuiop " $= . 0 OK (longueurs différentes)
```

(") --- str

Mot système compilé par le mot "guillemet" (voir ce mot) lorsqu'il est utilisé dans la définition d'un mot X pour définir une chaîne de caractères en tant que "littéral". Il est suivi par un octet de longueur, et par les caractères de la chaîne.
Son action lors de l'exécution du mot X est de déposer sur la pile les paramètres de la chaîne.

(NBR) adr --- d

Mot système appelé par le mot NUMBER. L'action de (NBR) est de convertir en nombre double longueur la chaîne de caractères commençant à l'adresse adr (l'octet en adr étant l'octet de longueur). (NBR) reconnait le signe "–" indiquant un nombre négatif, et le point indiquant un nombre double longueur.
Il ne reconnaît pas le signe "+", de sorte que +12 n'est pas considéré comme un nombre. Les chaînes contenant un symbole autre que "–", ".", ou des chiffres provoquent un message "UK ERROR", signifiant que (NBR) a rencontré une expression inconnue ou inconvertible.
Après exécution de (NBR), la variable DPL contient –1 si la chaîne à convertir correspondait à un nombre 16 bits (c'est-à-dire ne contenait pas de point), et le nombre de caractères situés après le point dans le cas d'un nombre 32 bits.
Attention, même dans le cas d'un nombre 16 bits, le résultat de (NBR) est un nombre double longueur.

```
" 345.678"  DROP 1- (NBR) D. DPL ?  345678 3 OK
" 1234567"  DROP 1- (NBR) D. DPL ?  1234567 -1 OK
```

Dans ce dernier cas, DPL contient –1. Cela semble indiquer un nombre 16 bits, mais (NBR) donne quand même un résultat correct. Il y a là une erreur de manipulation de l'opérateur, qui aurait dû taper un point pour indiquer l'entrée d'un nombre 32-bits.
Cette erreur aurait amené un résultat faux si le nombre 1234567 était entré au clavier (sans guillemets), et avait été converti (par l'intermédiaire de (NBR)) par l'interpréteur. Il faut savoir, en effet, que l'interpréteur teste DPL pour déterminer si le nombre tapé est 16 ou 32 bits, et s'il trouve la valeur –1, transforme le résultat de (NBR) en nombre 16 bits.

-TRAILING str1 --- str2

Supprime les blancs à la fin de la chaîne de caractères (str1). Le résultat est la chaîne (str2). L'adresse de début de chaîne ne change donc pas, seule la longueur est modifiée.

```
" abc def    " TYPE    acb def    OK
" acb def    "  -TRAILING  TYPE acb def    OK
```

(l'espace après le "f" fait partie de message "OK"..., les autres ont été suprimés)

Ou encore : nombre de lignes écrites dans un écran de l'éditeur

```
: NBLIGNES      ( n1 --- n2  n1 = numéro d'écran. n2 = nombre de lignes)
        0 25 0 DO OVER I SWAP (LINE)
                    -TRAILING SWAPDROP    ( Donne la longueur de la ligne )
                    0= NOT +              ( Ajoute 0 si ligne d'espaces )
                LOOP SWAPDROP ;
```

< # ---

Initialise le processus de conversion-formattage d'un nombre 32 bits en chaîne de caractères (voir les mots #, #S et # >). En fait, la seule action de < # est d'initialiser la variable HLD.

?WORD c --- adr

Mot système qui lit le mot suivant du flot d'entrée, et le déplace à la fin du dictionnaire. Si le flot d'entrée est terminé, ?WORD envoie un message d'erreur "DL ERROR" (pas de délimiteur).
En fait, ?WORD appelle WORD et teste si le flot d'entrée est terminé.

ARRAY$ n1 n2 ---

Mot de définition utilisé sous la forme:

```
n1 n2 ARRAY$ < nom >
```

pour créer un tableau de n2 chaînes, chacune ayant au maximum n1 caractères, et dont le nom dans le dictionnaire est < nom >.
Par la suite, n < nom > dépose sur la pile les paramètres de la n-ième chaine (n-ième élément) du tableau < nom >.
Comme pour les tableaux numériques, les éléments sont numérotés de 0 à n2 - 1. Vous pouvez les numéroter à partir de 1, lors de

l'affectation, il vous faudra alors prévoir un élément de plus dans le tableau (élément Ø non utilisé).

Définition

```
8 7 ARRAY$ SEMAINE          ( 7 jours, 8 caractères par jour)
```

Affectation

```
" Lundi" Ø SEMAINE $!       ( Affecte le premier élément du tableau)
" Mardi" 1 SEMAINE $!        " Mercredi" 2 SEMAINE $! etc .
```

Utilisation du tableau SEMAINE

```
2 SEMAINE TYPE Mercredi OK
```

ASC str --- n

Dépose sur la pile le code ASCII du premier caractère de la chaîne de caractères dont les paramètres sont sur la pile.

```
" FORTH" ASC EMIT F OK
```

BASE --- adr

Dépose sur la pile l'adresse d'une variable contenant la base actuellement utilisée pour la conversion des nombres en entrée ou en sortie. Les bases possibles vont de 2 à 70.
Noter que "BASE ?" donnera toujours le résultat 1Ø. Pour connaître la base en cours, on peut par exemple fabriquer le mot suivant, qui affiche la base en décimal sans la modifier:

```
: BASE? BASE @ DUP DECIMAL . BASE ! ;
```

D'origine, FORTH connait les mots HEX et DECIMAL, mais vous pouvez fixer la base de votre choix. Pour certains problèmes, on pourra utiliser par exemple le mot :

```
: BINAIRE  2 BASE ! ;  ( Par exemple : 254 BINAIRE . 11111110 OK)
```

Pour visualiser l'ensemble des symboles d'une base :

```
: SYMBOLES BASE @ Ø DO I . LOOP ;
HEX SYMBOLES Ø 1 2 3 4 5 6 7 8 9 A B C D E F OK
```

CHARTAB n ---

Mot de définition servant à créer et à utiliser une table de n caractères graphiques (n doit être compris entre Ø et 1Ø2).

Définition d'une table

n CHARTAB < nom > crée un tableau destiné à ranger n caractères, qui sera rempli en utilisant DEFGR, et utilisé par EMIT, par exemple.

Le nombre maximum de caractères définissable dans une table (valeur maximum du nombre n) est 1Ø2. Pour l'explication de ce nombre curieux, voir le mot DEFGR.

Utilisation d'une table

Il suffit ensuite de taper < nom > pour sélectionner la table < nom > .

Les codes ASCII à partir de 154 sont alors considérés comme les codes des caractères de la table < nom > (voir DEFGR pour la relation code-caractère). On peut ainsi disposer de plusieurs jeux de caractères graphiques simultanément.

Exemple :

45 CHARTAB XXX	Réserve de la place en mémoire pour une table XXX pouvant contenir au maximum 45 caractères.
XXX	Sélectionne cette table comme actuellement utilisée (pour remplir la table, ou pour l'utiliser). Remplissage de la table (définition des caractères; voir DEFGR). Eventuellement, définition ou utilisation d'autres tables...
XXX	Pour utiliser la table, ou pour redéfinir un caractère existant. Si on n'a pas sélectionné d'autre table depuis le précédent XXX, cette manoeuvre est inutile.
154 EMIT	Affiche le caractère défini avec le numéro 154 (en principe le premier de la table).
163 EMIT	Affiche le caractère défini avec le numéro 163.

Il est possible aussi de placer des caractères graphiques dans des chaînes de caractères. Ces chaînes seront utilisables par TYPE, et donneront des affichages différents suivant la table de caractères utilisée.

CHR$ n --- adr 1

Convertit un code ASCII en une chaîne *temporaire* de un caractère.

```
65 CHR$ TYPE A OK
```

Attention, le résultat de CHR$ est une chaîne de caractères, et non un caractère. Pour l'afficher, il convient d'utiliser TYPE, et non pas EMIT.
La raison d'être de ce mot est de permettre l'utilisation des mots concernant les chaînes de caractères (concaténation, etc.). Il réalise en quelque sorte l'opération inverse du mot ASC.

CONVERT d1 adr1 --- d2 adr2

Mot système. Convertit la chaîne de caractères commencant à adr1 + 1 (adr1 doit contenir l'octet de longueur) en nombre 32 bits, compte tenu de la base. Chacun des chiffres de la chaîne est converti par DIGIT, et multiplié par la puissance convenable de la base. La valeur ainsi obtenue est ajoutée à d1, et le total est laissé sur la pile à la fin de l'opération de conversion (c'est d2).
adr2 est l'adresse du premier caractère non convertible. Si la chaîne contient un point décimal, sa position est sauvegardée dans la variable DPL.
CONVERT est le mot de base du système pour les conversions, appelé par (NBR) ou VAL. En principe, l'utilisateur se sert plutôt des mots de niveau plus élevé, comme INPUT ou LINPUT.

COUNT adr --- adr+1 n

Mot système. A partir d'une chaîne de caractères commençant en adr (l'octet de longueur étant à l'adresse adr), COUNT donne des paramètres utilisables par les mots habituels qui opèrent sur les chaînes, c'est-à-dire l'adresse du début effectif de la chaine (adr + 1) et sa longueur (n).

DECIMAL ---

Fixe la base à dix. Voir BASE.
C'est le moyen le plus simple de revenir en base dix (si, étant dans une base x quelconque, on tape "1Ø BASE !" on ne modifiera pas la base car en base x, x s'écrit justement 1Ø).

DEFGR n1 n2 ... n8 n ---

Définit le caractère graphique numéro n dans la dernière table graphique positionnée.
Les caractères sont composés de huit segments horizontaux de huit points chacun. Ces points peuvent être individuellement allumés (couleur "point", définie par INK COLOR) ou éteints (couleur "fond", définie par PAPER COLOR). Ils sont représentés en mémoire par des Ø et des 1 (1 pour "allumé", Ø pour "éteint"). Un caractère est donc défini par huit octets en mémoire (huit fois huit bits).
Les nombres n1, n2, ... n8 sont les valeurs numériques (de Ø à 255 en base 10) des huit octets définissant le caractère de code ASCII *n* (n1 correspond au segment supérieur).
n est le numéro sous lequel ce caractère pourra être utilisé par EMIT. Les numéros possibles vont de 154 à 255 (les codes de Ø à 127 sont réservés pour l'affichage habituel. Les codes de 128 à 153 servent à coder les caractères spéciaux, comme le ç, les voyelles accentuées, et certains caractères spéciaux étrangers. Voir la table des codes ASCII étendus, en annexe).
Une table de un caractère ne comportera que le caractère de code ASCII 154. Une table de cinq caractères ne pourra contenir que les caractères de codes entre 154 et 158.
Il est possible de définir (ou d'utiliser) des caractères graphiques dans n'importe quel ordre, et pour plusieurs tables alternativement. On peut avoir autant de tables qu'on veut en mémoire, mais une seule active (sélectionnée) à la fois.

Exemple :

```
4 CHARTAB CARTES
```

crée une table de caractères graphiques nommée CARTES. Cette table peut contenir quatre caractères, numérotés (codes ASCII étendus) de 154 à 157.
(Autres commandes, ou rien du tout. Il n'est pas indispensable d'utiliser immédiatement la table CARTES. On peut même définir d'autres tables.)

```
HEX CARTES Ø 36 7F 7F 3E 1C 8 8 DECIMAL 156 DEFGR
```

Fabrique le troisième caractère de la table CARTES. Si on n'a pas utilisé d'autre table entretemps, inutile de préciser "CARTES".

Par la suite...

CARTES DECIMAL 156 EMIT Affiche le caractère défini ci-dessus, un coeur.
Ici encore, il n'est nécessaire de sélectionner la table "CARTES" que si cela n'a pas déjà été fait.

DIGIT c n1 --- (n2) cond

Mot système. En utilisant la base n1, convertit le caractère de code ASCII *c* en l'équivalent binaire du chiffre qu'il représente (compte tenu de la base en cours), accompagné d'une valeur de vérité 1, pour annoncer que la conversion s'est bien passé. Si la conversion n'est pas licite, DIGIT ne laisse sur la pile que la valeur de vérité Ø (la conversion a échoué).
Ce mot est avant tout un mot système, utilisé par CONVERT.

```
: CHIFFRE?          ( n1 n2 ---   Affiche n1 dans la base dix si c'est un)
                    ( chiffre en base n2. Sinon, message d'erreur.)
            DIGIT IF BASE @ SWAP DECIMAL . BASE !
                    ELSE . " Symbole invalide "* THEN ;
" A" ASC 7 CHIFFRE? Symbole invalide OK
    ( En base 7, les seuls symboles numériques sont Ø, 1, 2, 3, 4, 5, 6 )
" A" ASC 16 CHIFFRE? 1Ø OK
    ( En base 16, le "chiffre" A représente le nombre "dix"* )
```

ENCLOSE adr c --- adr n1 n2 n3

Mot système. ENCLOSE est la primitive appelée par WORD pour obtenir le prochain mot délimité par le caractère c (le plus souvent l'espace), à partir de l'adresse adr. Le caractère de code ASCII Ø est toujours considéré comme délimiteur inconditionnel.
ENCLOSE laisse sur la pile, outre l'adresse de début du mot, les offsets (adresses relatives à adr) n1, n2, n3, qui ont les significations suivantes :
n1 est l'offset du premier caractère du mot trouvé (premier caractère autre que c).
n2 est l'offset du premier délimiteur qui suit le mot trouvé.
n3 est l'offset du premier caractère non examiné (celui où débutera la prochaine recherche).

Vous pouvez vous en servir pour lire des mots directement dans des blocs, par exemple pour une gestion de fichier séquentiel en mémoire. Exemple : si les enregistrements de votre fichier utilisent le caractère "%" (code ASCII 37) comme séparateur, et en appelant > FICH une variable contenant l'adresse actuelle dans le fichier...

```
: LIRE-FICHE    > FICH    @    37    ENCLOSE
        > FICH    +!    ( pour la prochaine recherche )
        OVER    -    >R    +    R>    ;
```

laisse sur la pile les paramètres de la chaîne de caractères qui constitue l'enregistrement suivant...

HEX ---

Fixe la base à seize (système hexadécimal). Voir BASE.

```
DECIMAL 11 HEX . B OK          10 1-  F OK
```

Un "truc" pratique: "10 1-" vous donnera toujours le dernier symbole de la base utilisée (si un jour vous ne vous souvenez plus de la base en cours, tapez "10 1- ." ...).

HLD --- adr

Variable contenant l'adresse du dernier chiffre (code ASCII du caractère correspondant) lors d'une conversion numérique par < # ... # >. La zone où est rangée la chaîne de caractères exprimant un nombre débute à PAD – 1, et s'étend vers les adresses décroissantes. Ainsi, on trouve en PAD – 1 le chiffre des "unités", en PAD – 2 le chiffre des "dizaines" (en base dix ; en base seize, il s'agirait plutôt de "seizaines"...), et ainsi de suite. En fin de conversion, le chiffre le plus à gauche de la chaîne de caractères représentant le nombre est à l'adresse contenue dans la variable HLD.
Cette variable sert au système pour tous les affichages numériques, et l'utilisateur n'a théoriquement pas à s'en servir, sauf bien sûr s'il désire écrire lui-même ses mots de conversion-affichage numériques (flottant par exemple)

HOLD car ---

Insère un caractère ASCII dans une conversion-formatage d'un nombre 32 bits en chaîne de caractères utilisant < # et # >. Voir #.

INSTR str1 str2 --- n

Recherche la première occurence de la chaîne str1 dans la chaîne str2. *n* est la position du premier caractère de str1 pour cette occurence, et vaut Ø si str1 n'est pas une sous-chaîne de str2.

```
12 STRING MOT " Structure" MOT $!
" truc" MOT INSTR . 2 OK
" machin" MOT INSTR Ø OK
```

LEFT$ str1 n --- str2

Extrait la sous-chaîne constituée des *n* caractères de gauche de (str1). En fait, LEFT$ ne fait que modifier SUR LA PILE les paramètres de (str1); (str1) n'est pas modifiée, et il n'y a pas création d'une nouvelle chaîne. Pour modifier une chaîne, voir SUB$!.
La longueur de (str2) est inférieure ou égale à la fois à n et à la longueur de (str1).

```
2Ø STRING XX " définition" XX $!
XX 4 LEFT$ TYPE défi OK
XX 1ØØØ LEFT$ DDUP LEN 1Ø OK
TYPE définition OK
```

LEN str --- n

Donne la longueur de la chaîne (str).

```
" abcde" LEN . 5 OK
2Ø STRING XX " abcde" XX $! XX LEN . 5 OK
" abcde" XX $! XX LEN . 6 OK
```

MID$ str1 n1 n2 --- str2

Extrait la sous-chaîne de (str1) composée de n2 caractères à partir du caractère n1.

```
3Ø STRING XX " nombre 16 bits" XX $!
XX 2 5 MID$ TYPE ombre OK
```

MLEN str --- n

Donne la longueur maximum de la chaîne (str). Cette commande n'est licite que pour les chaînes définies comme variables (une chaîne temporaire n'a pas de longueur maximum, elle ne connaît que sa valeur courante...).

```
10 STRING AA " azerty" AA $!
AA MLEN . 10 OK
AA TYPE azerty OK AA LEN . 6 OK
```

NUMBER adr --- d

Mot de conversion de la chaîne de caractères (située à l'adresse adr) en nombre. Dans le FORTH de base, NUMBER exécute le mot (NBR), mais il est possible de changer cet état de choses, ce qui permet d'introduire d'autres types de nombres. Voir pour cela l'annexe concernant l'utilisation de la vectorisation (et le mot SYSVEC).

PAD --- adr

Laisse sur la pile l'adresse du début de la zone temporaire utilisée par le traitement des chaînes de caractères. C'est là que sont rangées entre autres toutes les chaînes définies par le mot "guillemet" en mode interactif, les chaînes entrées en réponse à des commandes du type INPUT ou INPUT$, etc.
L'octet à l'adresse donnée par PAD contient la longueur utilisée de la zone temporaire. Cette longueur est en principe limitée à 255 octets, si on veut être certain de ne pas empiéter sur le reste de la mémoire. En général, la zone temporaire est suivie par une zone mémoire disponible, mais si on fixe un maximum de buffers elle peut n'être effectivement que de 255 octets. Voir à ce sujet l'annexe E, relative à l'organisation mémoire.

```
" ABCDE" OK
PAD C@ . 5 OK ( longueur de la chaîne rangée en zone temporaire )
PAD COUNT TYPE ABCDE OK
```

Cette zone est située 84 octets après la fin actuelle du dictionnaire, et change donc lorsqu'on crée un nouveau mot, ou lorsqu'on en supprime (FORGET).

```
PAD . 27179 OK : XX ; PAD . 27188 OK
```

PAD marque aussi la fin de la zone de conversion des nombres en chaînes de caractères (zone de longueur variable, suivant le nombre).

```
PAD 4 - 4 TYPE 7188 OK  ( 4 derniers chiffres de 27188... Voir ci-dessus.)
```

RIGHT$ str1 n --- str2

Analogue à LEFT$. (Remplacer gauche par droite.)

```
20 STRING XX " arithmétique" XX $!
XX 5 RIGHT$ TYPE tique OK
```

SIGN n ---

Utilisé entre < # et # > (conversion avec formatage d'un nombre 32 bits en chaîne de caractères) pour insérer le signe " – " (ASCII 45) si le nombre *n* est négatif.

```
: .(-3)          ( d --- str )
    DUP  >R      ( Pour mettre le signe en réserve )
    DABS  <#  #  #  #  44 HOLD  #S  R>  SIGN  #>  ;
```

Puis :

```
1234.   .(-3)  TYPE  1,234 OK
-123.45  .(-3)  TYPE  -12,345 OK
```

STR$ d --- str

Conversion "standard" d'un nombre 32 bits en chaîne de caractères. En fait, STR$ est défini de la manière suivante :

```
: STR$  SWAP  OVER  DABS  <#  #S  ROT  SIGN  #> ;
```

On a utilisé ici "SWAP OVER" pour sauvegarder la partie de poids fort de *d* (qui donne le signe). Voir aussi l'exemple .(–3) du mot SIGN pour une autre méthode ("DUP > R", qui utilise la pile des retours). Voir aussi USTR$.

STRING n ---

Mot de définition utilisé sous la forme :

```
n STRING < nom >
```

pour créer une variable chaîne de caractères, dont le nom dans le dictionnaire est < nom > , et permettant de ranger des chaînes de lon-

gueur inférieure ou égale à n (n est la longueur maximum).
Lorsque par la suite on exécutera le mot < nom >, les paramètres de cette chaîne (adresse de début de rangement et longueur courante) apparaîtront sur la pile.

```
4  STRING  NOM      " Alexandre"  NOM   $!
NOM  LEN  .  4 OK   ( longueur maximum )
NOM  TYPE  Alex OK  ( "Alexandre" a été tronqué à quatre caractères..  )
15  STRING  MOT      " Il fait beau"  MOT   $!
MOT LEN TYPE  12 OK     MOT  MLEN  TYPE   15 OK
```

D'une manière générale, le pf de < nom > contient un octet de longueur maximum (contenant le nombre 15 dans le dernier exemple), un octet de longueur courante contenant le nombre 12), et n octets (ici, n = 15) contenant les caractères de la chaîne.

SUB$! str1 str2 ---

L'action de SUB$! est très voisine de celle de $!, à cela près que SUB$! ne modifie pas la longueur de str2. SUB$! range simplement (str1) dans la zone mémoire de paramètres str2, sans faire de vérifications. Il est donc dangereux d'utiliser SUB$! à l'aveuglette. Un bon moyen est de définir systématiquement str2 en n'utilisant que LEFT$, MID$, ou RIGHT$.

```
33  STRING  PHRASE      " Tableau de nombres"  PHRASE   $!
" chaînes"  PHRASE   7  RIGHT$   SUB$!
PHRASE   TYPE  Tableau de chaînes OK
```

TEXT c ---

Mot système. TEXT lit le mot suivant du flot d'entrée (délimité par le caractère c), et le recopie en PAD, l'octet en PAD étant l'octet de longueur du mot.

USTR$ ud --- str

Conversion standard du nombre 32 bits ud, considéré comme non signé, en une chaîne de caractères dont les paramètres (str) sont laissés sur la pile.
En fait, la définition de USTR$ est simplement:

```
: USTR$   < #   #S   # >   ;
```

VAL str --- d

Convertit la chaîne (str), composée de chiffres relativement à la base en cours d'utilisation, en nombre 32 bits dont la représentation binaire est laissée sur la pile. La conversion utilise le mot CONVERT, donc s'arrête au premier caractère non valide compte tenu de la base.

```
10 STRING NBRE      " 12AGB" NBRE $!
DECIMAL NBRE VAL D.  12 OK
HEX NBRE VAL D.  12A OK
17 BASE ! NBRE VAL D.  12AGB OK
```

VAL est utilisé par LINPUT et INPUT.

WORD c --- adr

Lit le mot suivant du flot d'entrée, et le copie à la fin du dictionnaire (l'adresse laissée sur la pile est en fait l'adresse que donne HERE, c'est celle du premier octet disponible à la fin du dictionnaire. Cet octet contient la longueur du mot lu). Le dictionnaire n'est pas augmenté pour autant, mais il est prêt à l'être...

WORD appelle ENCLOSE, l'adresse de début de la recherche étant calculée grêce au contenu des variables BLK et >IN.

Si BLK contient 0, le flot d'entrée est le clavier (adresse de base donnée par "TIB @"), sinon BLK contient le numéro d'un BLOCK (l'adresse de base du flot d'entrée sera alors donnée par l'expression "BLK @ BLOCK").

>IN est l'offset par rapport au début du flot d'entrée, à ajouter à l'adresse de base pour obtenir l'adresse du début de la recherche.

WORD est le mot système utilisé par l'interpréteur externe pour lire le flot d'entrée, mais vous pouvez aussi vous en servir pour définir des mots utilisant des paramètres suivant le mot (travailler en notation préfixée).

Exemple : mot ASCII, donnant le code ASCII du mot suivant du flot d'entrée...

```
: ASCII          ( --- n )
    BL  WORD  1+  C@  ;
ASCII  A  .  65 OK     ( Pas de message "Mot inconnu", bien que "A" ne soit )
                       ( pas un mot du dictionnaire)
```

Note : si vous voulez employer ASCII dans une définition, il convient de bien saisir la notion de "flot d'entrée". WORD lit le mot suivant du flot d'entrée *au moment où il s'exécute.*

Autrement dit, la locution "ASCII A" provoquera un message d'erreur en compilation si le mot ASCII n'est pas déclaré comme immédiat... A moins de l'utiliser entre crochets, comme ci-dessous. Par contre, vous pouvez l'utiliser par exemple de la manière suivante:

```
: SEXE?        ( --- cond )
  ASCII        (Lira le code ASCII du "mot" suivant lors de l'exécution )
  CASE         [ ASCII 1 ] OF  ." Sexe masculin"   ENDOF
               [ ASCII 2 ] OF  ." Sexe féminin"    ENDOF
               ." Erreur sur le numéro de S.S.'
  ENDCASE ;
```

Puis :

```
SEXE?  1460759620006   Sexe masculin OK
SEXE?  2480762128035   Sexe féminin OK
SEXE?  Dupont   Erreur sur le numéro de S.S. OK
```

Ce genre d'exemple serait surtout intéressant si le flot d'entrée était un block en mémoire ou sur disque (gestion de fichier). Il suffit pour cela de positionner correctement les variables BLK et >IN, la définition de SEXE? n'a pas à être modifiée...

Chapitre 6

Graphique et couleur

L'affichage de votre ordinateur est entièrement graphique. Cela signifie en particulier que même lorsque vous écrivez du texte à l'écran, ce texte est affiché en tant que séquence de caractères graphiques. Ce système, au prix d'un affichage un peu plus lent, donne la possibilité de mélanger textes et dessins, et de définir vos propres jeux de caractères affichables.

Les mots détaillés ici ne sont en général pas standards, puisqu'ils sont écrits spécifiquement pour votre appareil. Nous n'avons pas précisé ici les mots habituels qui gèrent les Entrées-Sorties du clavier et de l'écran (par exemple EMIT, KEY, TYPE, SPACE, etc.). Vous les trouverez dans la section consacrée aux entrées-sorties (chapitre 9).

La gestion des couleurs se fait de la manière suivante : chaque point de l'écran correspond à un des bits d'un octet. Cet octet se voit attribuer deux couleurs, une couleur de fond (PAPER) et une couleur de point (INK), et s'affiche comme un petit segment horizontal de huit points. Les bits à Ø donnent des points de la couleur du fond, et les bits à 1, des points de la couleur de point.
Il est donc impossible physiquement d'avoir trois couleurs par segment.

Les dessins s'obtiennent alors en précisant quels points sont de la couleur « fond », et quels points sont de la couleur « point ». Cette distinction fond-point est totalement arbitraire, et les deux sont complètement interchangeables. Vous pouvez par exemple dessiner des points en couleur "fond" sur un fond de couleur « point ». Les mots PAPER et INK agissent sur toutes les commandes ultérieures.
Par exemple, GCOLOR positionnera la couleur « fond » ou « point » pour les commandes graphiques ultérieures, puis PSET dessinera un point "fond" ou "point", suivant que vous êtes en mode « fond » ou « point »...
Par ailleurs, les attributs de couleur peuvent être valables pour tout l'écran (attributs « globaux », positionnés par le mot GLOBAL), ou laisser l'écran tel qu'il est et ne prendre effet que pour les affichages ultérieurs (attributs « locaux », positionnés par le mot LOCAL). Cela n'est valable que pour les mots concernant l'utilisation de texte, les mots purement graphiques se comportent toujours de manière « locale ».

Les couleurs sont codées par des nombres de Ø à 7, dans l'ordre: noir - rouge - vert - jaune - bleu - magenta - cyan - blanc.
Toutefois, FORTH les décrit par leur nom pour vous faciliter le travail. Si par ailleurs vous possédez un TO7-70 ou un MO5, vous disposerez de deux jeux de huit couleurs, et vous pourrez passer d'une palette à l'autre par les mots PALE et DARK. La couleur effectivement affichée peut alors dépendre de votre système, et du réglage de votre téléviseur...

En ce qui concerne les mots qui permettent de dessiner à l'écran, ils utilisent les « coordonnées-écran » des points à dessiner. Chaque point est repéré par deux nombres :
Une coordonnée horizontale (abscisse) de Ø à 319, de la gauche vers la droite.
Une coordonnée verticale (ordonnée) de Ø à 199, du haut vers le bas.
Ainsi, le coin en haut à gauche de l'écran est le point (Ø,Ø), et le coin en bas à droite est le point (319,199).

BLACK --- n

Une constante qui dépose sur la pile le code de la couleur noire (Ø).

BLUE --- n

Une constante, qui dépose sur la pile le code de la couleur bleue (4).

BOX n1 n2 n3 n4 ---

Dessine un rectangle à cotés horizontaux et verticaux. (n1,n2) et (n3,n4) sont des sommets opposés. Voir PSET pour les couleurs utilisées. Le dernier point tracé par BOX est le point (n3,n4) (pour utilisation par les mots en « TO », comme BOXTO, BOXFTO, LINETO...

```
INK 1 1 318 198 BOX
```

Fait un cadre autour de l'écran (en couleur « point »).

BOXF n1 n2 n3 n4 ---

Comme pour BOX, mais remplit le rectangle compte tenu du positionnement de la couleur (voir PSET pour la couleur utilisée). Le dernier point tracé par BOXF est le point (n3,n4).

```
PAPER Ø Ø 15 2Ø BOXF
```

Dessine un rectangle plein, en couleur « fond ».
Attention, si le fond était déjà rempli d'une autre couleur « fond », la commande :

```
PAPER 2 Ø 1Ø 2Ø BOXF
```

aurait la même action.
En effet, les points des colonnes Ø,1 (points d'ordonnées Ø ou 1) sont dans les mêmes octets que les points des colonnes 2 à 7, et leurs couleurs sont modifiées en même temps. Il va de même des points d'ordonnées 11 à 15.

Essayez : PAPER RED COLOR Ø Ø 15 2Ø BOXF
Suivi de : YELLOW COLOR Ø Ø 1Ø 2Ø BOXF

BOXFTO n1 n2 --

Analogue à BOXF, mais le rectangle tracé a pour sommets opposés d'une part le dernier point dessiné auparavant, d'autre part le point (n1,n2).
Le dernier point dessiné par BOXFTO est le point (n1, n2).

```
10 10 PSET 200 100 BOXFTO
```
Puis :
```
220 120 BOXFTO
                    ( Le dernier point dessiné était le point 200,100)
```

BOXPEN n1 n2 n3 n4 n ---

Sert à définir une zone écran (rectangle) en mode graphique pour utilisation ultérieure par ONPEN (utilisation du light-pen). La zone définie est un rectangle de sommets opposés (n1,n2) et (n3,n4), et a le numéro n (lu par ONPEN).
n est un nombre entre 0 et 7. Voir PEN pour plus d'explications.
Les zones de pointage ainsi définies sont commodes à utiliser en liaison avec CASE ou GODO.

```
10 50 30 100 3 BOXPEN
```
Définit comme zone 3 le rectangle de diagonale (10,50) – (30,100).
Pour utilisation, voir ONPEN.
Il est souvent intéressant de colorier ces zones. Cela peut se faire comme suit (si on veut éviter d'avoir à se souvenir des nombres):

```
10 50 30 100    DOVER DOVER BOXF    3 BOXPEN
```

BOXTO n1 n2 ---

Analogue à BOX, mais le rectangle dessiné a pour sommets opposés d'une part le dernier point tracé, d'autre part le point (n1,n2). Le dernier point tracé par BOXTO est le point (n1,n2).

```
: POINT-HASARD  ( --- x y )  320  RND  200  RND  ;
: RECTANGLES    ( --- )
    POINT-HASARD  PSET
    BEGIN  POINT-HASARD  BOXTO  PTRIG
    UNTIL  ;
```

CLS ---

Efface l'écran (ou seulement la fenêtre de travail, si on en a défini une) et positionne le curseur dans le coin en haut à gauche.

COLOR n ---

Positionne la couleur du texte.
Agit sur les caractères ou sur le fond suivant le dernier mot positionné, INK ou PAPER. COLOR n'agit pas sur la couleur du graphique (voir GCOLOR).
En outre, le mot COLOR aura un effet immédiat et global, ou local et retardé, suivant qu'on est en mode LOCAL ou en mode GLOBAL (voir ces mots).
n représente le code de la couleur. Il est plus facile et plus lisible d'utiliser les constantes BLACK, RED, GREEN, YELLOW, BLUE, MAGENTA, CYAN et WHITE, mais "BLACK COLOR" et "Ø COLOR" auront le même effet.

```
LOCAL  INK            ( positionnement standard)
BLACK  COLOR  OK      ( OK est écrit en noir)
RED    COLOR  OK      ( RED COLOR est en noir, et OK en rouge )
```

Voir aussi les mots LOCAL, GLOBAL, PAPER, et INK pour comprendre complètement le fonctionnement du mot COLOR.

COLOROFF ---

Les attributions de couleur ultérieures n'ont pas d'effet tant que l'action de COLOROFF n'est pas annulée par COLORON (voir ce mot).
En mode COLOROFF, les couleurs utilisées pour l'affichage sont les couleurs des caractères déjà présents à l'écran. Si cet affichage est en plusieurs couleurs, on verra alors le texte défiler en changeant de couleur.

```
: COLORIAGE  (En mode COLORON)
    LOCAL PAPER RED COLOR INK YELLOW COLOR
    25 Ø DO Ø I LOCATE 1Ø SPACES LOOP
    PAPER WHITE COLOR INK BLACK COLOR
    25 Ø DO 2Ø I LOCATE 1Ø SPACES LOOP
    ;              ( ... Et un peu plus si vous voulez )
```

Et enfin : CLS COLORIAGE COLOROFF VLIST (En une seule commande...)
La liste des mots du dictionnaire se déroule en prenant les couleurs définies par les instructions précédentes (une autre manière de définir des zones de couleurs différentes, plus rapide mais limitée à des tranches horizontales d'écran, est d'utiliser WINDOW et des attributs de couleur globaux).

COLORON ---

Option de base du FORTH. COLORON sert à mettre fin au mode COLOROFF, c'est à dire qu'il rétablit la prise en compte par l'affichage du positionnement en cours des couleurs (couleurs, mode local ou global, mode point ou fond).

CSRLIN --- n

Laisse sur la pile le numéro de la ligne où est positionné le curseur (entre Ø et 24), même s'il est invisible (voir CSROFF et CSRON). Voir le mot POS (analogue à CSRLIN, mais pour les colonnes).
Exemple : écrire en *n* lignes, sans utiliser le mot WINDOW.

```
VARIABLE LGN          ( Nombre de lignes )
: LIT-ECRIT           ( n ---  écrire en n lignes)
    1- LGN ! HOME
    BEGIN KEY DUP 13 =
     IF CR DROP ELSE EMIT THEN
     CSRLIN LGN    >
     IF HOME THEN PTRIG
    UNTIL ;
```

CSROFF ---

Rend le curseur invisible. Cette option est souvent intéressante à l'occasion de graphiques, où il n'est pas agréable de voir le curseur clignoter à un endroit quelconque de l'écran.
Exemple : initialisation de l'écran avant de dessiner

```
: INIT-ECRAN  WHITE  PAPER  GLOBAL  COLOR  CSROFF  CLS  ;
        ( La couleur graphique est en général fixée au fur et à mesure)
```

CSRON ---

Rend le curseur visible. Ce mot annule l'option CSROFF. Il est fréquent d'utiliser CSROFF au début d'une phase graphique, et CSRON lorsqu'on revient en mode texte (surtout lorsqu'on veut des entrées du clavier).

CYAN --- n

Une constante, qui dépose sur la pile le code ASCII de la couleur cyan (6).

DARK ---

Sur le MO-5 et le TO7-70, permet d'obtenir la palette des couleurs saturées. Voir aussi le mot PALE (couleurs non saturées).

```
DARK BLUE COLOR
```

FRAME n ---

Positionne la couleur du pourtour de l'écran à la valeur *n*.
Exemple : faire clignoter le pourtour de l'écran

```
: CLIGNOTE
    20 0 DO I 2 MOD FRAME    ( Alternativement noir et rouge )
            100  WAIT  LOOP
     CYAN FRAME ;
```

Vous pouvez ensuite, dans un autre mot, utiliser :

```
... ERREUR IF CLIGNOTE THEN ...
```

GCOLOR n ---

Positionne la couleur en mode graphique. Agit sur les caractères ou sur le fond suivant le dernier mot positionné, INK ou PAPER. *n* représente le code de la couleur. Il est plus facile et plus lisible d'utiliser les constantes BLACK, RED, GREEN, YELLOW, BLUE, MAGENTA, CYAN et WHITE, qui valent respectivement 0, 1, 2, 3, 4, 5, 6 et 7.

```
PAPER RED GCOLOR 10 20 300 100 BOXF (Boîte en couleur "fond")
INK YELLOW GCOLOR 60 80 BOXFTO (Boîte en couleur "point")
```

GLOBAL ---

Option pour le mot COLOR (GCOLOR fonctionne toujours en mode LOCAL). En mode GLOBAL, les attributions de couleur prennent effet immédiatement, et concernent tout l'écran (en fait toute la fenêtre de travail). Les couleurs des caractères déjà affichés sont modifiées instantanément.
Cette option peut être supprimée par le mot LOCAL.

```
VLIST                    ( Pour remplir l'écran )
GLOBAL INK BLACK COLOR   ( Tous les caractères en noir )
PAPER YELLOW COLOR       ( Fond tout jaune )
Ø 15 WINDOW GREEN COLOR  ( Fond vert dans la partie supérieure de l'écran )
LOCAL INK MAGENTA COLOR  ( Caractères violets, mais les caractères déjà )
                         ( affichés ne sont pas modifiés ).
```

GREEN --- n

Laisse sur la pile le code de la couleur vert (2)

HOME ---

Positionne le curseur en haut à gauche de l'écran (ou de la fenêtre en cours d'utilisation), sans remise à zéro de ce dernier (contrairement à CLS).
HOME sert aussi à revenir en début de fenêtre après un affichage en dehors de celle-ci.

```
10 15 WINDOW 20 Ø LOCATE . " coucou " HOME
```

Ecrit "coucou" au début de la vingtième ligne, puis remet le curseur au début de la dixième ligne.

INITSCR ---

Mot de « confort », qui réinitialise l'écran. L'effet exact de INITSCR est le suivant: il remet les caractères en taille normale (Ø Ø SIZE), réaffiche le curseur (CSRON), se met en mode « COLORON », remet la fenêtre écran à sa taille maximum (Ø 24 WINDOW), revient au mode « défilement normal » (Ø SCROLL).
INITSCR ne modifie pas le positionnement des couleurs, de sorte que vous pouvez choisir vos couleurs préférées pour travailler.

INK ---

Indique que toutes les opérations de couleur se feront dorénavant sur les caractères (en mode texte) ou sur les points (mode graphique). INK est l'option de base, celle qui est en cours à la mise sous tension de votre ordinateur.
Voir aussi le mot PAPER.

INVCOLOR ---

Inverse les couleurs « fond » et « point » à partir de la position courante ou dans tout l'écran, suivant qu'on est en mode LOCAL ou GLOBAL.
Exemple : lire une touche en suggérant une réponse (mode local).

```
: LITOUCHE    ( n --- n = code ASCII de la touche suggérée )
    INVCOLOR  EMIT  INVCOLOR  8  EMIT  KEY  DUP  EMIT  ;
```

Puis : `49 LITOUCHE ( Attend la frappe d'une touche, et suggère un 1 )`
Ou: `" A" ASC LITOUCHE ( au lieu de 65 LITOUCHE ; suggère un A )`

Autre exemple :

```
GLOBAL  INVCOLOR  ( passe en inversion vidéo dans la fenêtre )
```

LINE n1 n2 n3 n4 ---

Trace une ligne droite entre les points de coordonnées (n1,n2) et (n3,n4). Le dernier point tracé par LINE est le point (n3,n4) (pour utilisation par les mots LINETO, BOXTO,...)
Voir PSET pour l'utilisation des couleurs.

```
12  15  200  111  LINE
```

LINETO n1 n2 ---

Joint le dernier point positionné à l'écran avec le point (n1,n2). Le dernier point dessiné par LINETO est le point (n1,n2).
Voir PSET pour l'utilisation des couleurs.
Exemple : dessin au crayon optique

```
: INIT-ECRAN GLOBAL PAPER WHITE COLOR INK ;
: COULEUR 7 RND GCOLOR ;      ( Couleur au hasard sauf blanc )
: ZIK 12 RND 49 + NOTE ;
VARIABLE SAUT      ( Ø si suite du dessin sans saut, 1 s'il y a un saut )
: TRACE     ( x y --- )    SAUT @ IF PSET ELSE LINETO THEN ;
: DESSIN    ( --- )
    INIT-ECRAN Ø SAUT !
    INPUTPEN PSET     ( Appuyer le crayon optique pour commencer )
    BEGIN COULEUR ZIK
       INPEN DUP Ø< IF    DDROP 1
                          ELSE TRACE Ø THEN
       SAUT ! ?TERMINAL
    UNTIL          ( Arrêt si on tape une touche)
    CYAN PAPER GLOBAL COLOR CLS INITSCR INK LOCAL ;
```

Puis :

```
5 DUR DESSIN
```

Appuyez le crayon optique sur l'écran pour commencer, et dessinez en le promenant près de l'écran. Il suffit de l'écarter de l'écran pour arrêter le tracé, et de le rapprocher pour continuer ailleurs.

LOCAL ---

Mode pour COLOR. En mode LOCAL, les attributions de couleur n'agissent qu'à partir du moment de l'attribution, et ne concernent que les caractères effectivement affichés. Les couleurs des caractères déjà affichés ne sont pas modifiées.
LOCAL est l'option de base du FORTH, c'est celle qui est en effet à la mise sous tension. Voir GLOBAL pour l'option contraire.

LOCATE n1 n2 ---

Positionne le curseur en colonne n1, ligne n2. (n1 entre Ø et 39, n2 entre Ø et 24). LOCATE ne tient pas compte de la fenêtre.

```
30 4 LOCATE ." Coucou'
```

MAGENTA --- n

Laisse sur la pile le code de la couleur magenta (5).

MASK ---

Positionne le mode « masqué » sur la position courante.
Dans ce mode, les caractères sont affichés en couleur noire sur fond noir, de manière locale ou globale (suivant le mode LOCAL ou GLOBAL). Il est possible alors d'écrire à l'écran des mots qui seront invisibles tant qu'ils n'auront pas été démasqués par le mot NOMASK.
Utilisation possible : jeux à deux joueurs, du type « Deviner un nombre ou un mot ».
Le mot à deviner est entré au clavier en mode masqué, et est démasqué lorsqu'il a été trouvé.

NOMASK ---

Positionne le mode non masqué. Bien que ce soit simple à comprendre, ce mot agit de deux manières très différentes suivant qu'on est en mode local ou en mode global (voir les mots LOCAL et GLOBAL).
En mode LOCAL, NOMASK n'agit que sur les caractères affichés postérieurement. Autrement dit, les caractères précédemment masqués restent masqués. Ce mode sert surtout à revenir à un affichage normal, tout en conservant des parties masquées.
En mode GLOBAL NOMASK provoque un démasquage immédiat de toutes les zones masquées de la fenêtre de travail. Les parties masquées qui sont hors de la fenêtre ne sont pas concernées, ce qui permet des démasquages sélectifs.

```
VARIABLE  INCONNU
:  TEST-MASK ( --- )
   . " Donne un nombre :"  LOCAL  MASK  INPUT  NOMASK
   DROP  0  MAX  20  MIN  INCONNU  !  CR
   BEGIN  . " Quel est le nombre masqué ?"  INPUT DROP
     INCONNU  @  =  NOT
   WHILE  BELL  . " Raté !"  CR
   REPEAT
   BELL  BELL  BELL  . " Bravo !"  GLOBAL  NOMASK LOCAL ;
```

Essayez aussi des masquages et démasquages dans des fenêtres différentes...

PAGE ---

Positionne le mode "page" (sans défilement). Lorsque l'affichage arrive à la derniére ligne de l'écran, il se continue sur la première ligne, et les lignes déjà écrites restent à la même place (ne remontent pas d'un cran).
Ce mode est commode si vous voulez mélanger texte et graphique, ou si vous voulez par exemple gérer les entrées du clavier en utilisant un masque de saisie qui écrit sur la dernière ligne de l'écran (dans ce cas, le défilement perturberait les positions à l'écran et vous obligerait à recalculer les paramètres de vos "LOCATE").

```
PAGE VLIST
```

Voir le mot SCROLL, pour rétablir le défilement.

PALE ---

Mode pour la sélection des couleurs par COLOR ou GCOLOR, si vous disposez d'un TO7-70 ou d'un MO5. Les prochaines attributions de couleur seront non saturées (voir DARK).

```
PALE BLUE COLOR
```

PAPER ---

Sélectionne le « fond » pour la prochaine attribution de couleur, texte ou graphique, locale ou plein écran, et pour toutes les actions graphiques suivantes.

```
PAPER GREEN GCOLOR      ( fond vert)
```

PEN n1 n2 n ---

Sert à définir une zone écran (rectangle d'un caractère) en mode « texte » pour utilisation ultérieure par ONPEN (utilisation du crayon optique). La zone définie correspond à la position (n1,n2) d'un caractère en mode « texte » (n1 entre Ø et 39, n2 entre Ø et 24), et a le numéro n (lu par ONPEN).
Le numéro de zone n est un nombre de Ø à 7. Si vous tapez un nombre hors de ces limites, il sera ramené à la borne la plus proche. Par exemple, si vous essayez de définir une zone numéro −3, FORTH considèrera que vous avez défini la zone Ø (et l'ancienne zone Ø, si

elle existait, aura disparu, elle sera remplacée par celle que vous venez de définir). Les zones de Ø à 7 peuvent être indifféremment définies par PEN ou par BOXPEN.

```
5 1Ø DDUP 127 EMIT 6 PEN
```

Définit la zone correspondant au caractère (5,1Ø), représentée à l'écran par un pavé de couleur, comme étant la zone numéro 6.
Cette instruction équivaut à :

```
4Ø 8Ø 47 87 DOVER DOVER BOXF 6 BOXPEN
```

(On suppose que les couleurs « texte » et « graphique » soient les mêmes).

POINT n1 n2 --- n

Renvoie le statut du point de l'écran de coordonnées (n1,n2). Si c'est un point de couleur « point », n est le code de sa couleur. Si c'est un point de couleur « fond », n vaut $-x-1$, où x est le code de la couleur.

```
INK WHITE GCOLOR Ø Ø 16Ø 199 BOXF     ( Couleur blanche. x=7 )
PAPER GREEN GCOLOR 319 1 BOXFTO       ( Couleur verte. x=2 )
1ØØ 18Ø POINT . 7 OK     ( Couleur "point" blanche. x=n=7)
2ØØ 18Ø POINT . -3 OK    ( Couleur "fond" verte. x=2 donc n=-3 )
```

POS --- n

Laisse sur la pile le numéro de la colonne où est positionné le curseur (entre Ø et 39), même s'il n'est pas visible (voir CSRON et CSROFF). POS est l'analogue pour les colonnes du mot CSRLIN.
Exemple : écrire en *n* colonnes... (analogue à l'exemple de CSRLIN).

```
VARIABLE COLN
: LIT-ECRIT       ( n --- Ecrit en n colonnes )
    1- COLN ! CR
    BEGIN KEY DUP 13 =
       IF CR DROP ELSE EMIT THEN
       POS COLN @ >
       IF CR THEN PTRIG
    UNTIL ;
```

PSET n1 n2 ---

Positionne un point à l'écran en mode graphique, en colonne n1 (entre Ø et 319) et en ligne n2 (entre Ø et 199). Agit en mode "point" ou en mode "fond" suivant le dernier mot positionné INK ou PAPER. La couleur utilisée est celle définie par GCOLOR, sur le fond comme sur la forme.

```
PAPER GLOBAL BLACK COLOR WHITE GCOLOR
INK 160 100 PSET    ( Dessine un point à peu près au milieu de l'écran )
PAPER 160 102 PSET
```

Cette dernière commande dessine un segment, car tous les points de l'octet contenant le point (160,102) étaient en couleur « fond », donc sont modifiés par cette nouvelle attribution de couleur.

RED --- n

Laisse sur la pile le code de la couleur rouge (1)

SCREEN n1 n2 --- c

Laisse sur la pile le code ASCII du caractère présent à l'écran en ligne n1, colonne n2, n1 et n2 sont donc bien sûr des coordonnées en mode « texte » (n1 entre Ø et 39, n2 entre Ø et 24).
c vaut Ø si le caractère n'est pas reconnu.

```
15 24 WINDOW Ø Ø LOCATE ." ABCDEFGHIJKL" HOME
Ø Ø SCREEN . 65 OK      ( Lettre A )
1 Ø SCREEN . 66 OK      ( Lettre B )
```

et ainsi de suite... Il est possible d'analyser de cette manière tout l'écran du téléviseur.

SCROLL n ---

Positionne le mode « avec défilement », et avec deux vitesses possibles suivant la valeur de *n* (Ø pour vitesse normale, 1 pour vitesse lente).

```
1 SCROLL VLIST
Ø SCROLL VLIST
PAGE VLIST
```

SCROLL annule aussi le mode « page » (sans défilement).

Le défilement lent, associé à une grande taille des caractères (voir SIZE) et au mode « COLOROFF », permet des effets d'affichage très spectaculaires.

SIZE n1 n2 ---

Positionne la taille des caractères à l'affichage :

n1	n2	
Ø	Ø	Taille normale
Ø	1	Largeur double
1	Ø	Hauteur double
1	1	Taille double

Définissez d'abord des fenêtres multicolores (voir WINDOW), puis essayez:

```
1 1 SIZE COLOROFF 1 SCROLL VLIST
```

WHITE --- n

Laisse sur la pile le code de la couleur blanche (7)

WINDOW n1 n2 ---

Positionne la fenêtre d'affichage du texte. n1 est la première ligne, n2 la dernière. n1 et n2 doivent être compris entre Ø et 24, et n1 doit être inférieur à n2.
Le mode « global » n'agit à partir de là que dans la fenêtre, ainsi que la plupart des mots qui gèrent l'écran. Les exceptions sont FRAME (bien sûr), LOCATE, POINT et SCREEN.
Par exemple, HOME remet le curseur dans le coin supérieur droit de la fenêtre, d'où qu'il vienne, même du dehors de la fenêtre.

```
VLIST ...          ( Le curseur est sur la dernière ligne )
2 7 WINDOW PAPER RED COLOR CLS
                                   ( Seules les lignes 2 à 7 sont effacées...)
Ø 24 WINDOW        ( Revient à la fenêtre standard)
```

YELLOW --- n

Laisse sur la pile le code de la couleur jaune (3).

Chapitre 7
Musique

Il est possible de produire de la musique de deux façons : une note à la fois, en utilisant le mot NOTE, ou par l'intermédiaire d'une table musicale (voir MUSTAB), dans laquelle on compile une partition avant de l'exécuter par le mot PLAY.
Dans les deux cas, la gamme utilisée est la gamme tempérée (demi-tons égaux). Les notes sont codées par des nombres de 49 (do) à 61 (do+), le code augmentant d'une unité par demi-ton. Le code 48 représente un silence.
Il est aussi possible de fixer l'octave, le tempo, la durée de chacune des notes, et de jouer dans une certaine mesure sur l'enveloppe des sons produits.

Les mots dont le nom est en minuscules concernent la création de tables musicales. Leur action n'est donc pas immédiate, elle n'est effective que lorsqu'on joue la table.
Ceux dont le nom est en majuscules concernent l'utilisation habituelle, soit pour fixer des paramètres musicaux ou modifier des tables, soit pour produire effectivement de la musique note par note (voir NOTE) ou à partir d'une table musicale (voir PLAY).
Les mots MUSBUF, >MUS et >MUS+ sont des mots système, que vous pouvez toutefois utiliser pour modifier une table musicale.

. # ---

Augmente d'un demi-ton la dernière note compilée dans la table musicale en cours (voir MUSTAB). Permet de réaliser des dièses dans les partitions en table musicale. . # modifie seulement le code de la dernière note compilée, et ne modifie pas la taille actuelle de la table.

```
100 MUSTAB ZIK
ZIK do do .# re re .# mi fa fa .# sol sol .# la la .# si do+
PLAY          ( joue une gamme chromatique)
```

Notes :

— mi . # donnerait un fa, et si . # un do+.

— Aucun test de validité n'est effectué.

Evitez en particulier d'écrire "do+ . #", changez d'octave...

.b ---

Diminue d'un demi-ton la dernière note compilée dans la table musicale en cours. Permet d'écrire des "bémols" dans une partition en table musicale. .b modifie le code de la dernière note compilée, et n'influe pas sur la taille courante de la table.

La gamme utilisée est la gamme tempérée, ("re .b" a le même effet que "do . #").

```
100 MUSTAB ZIK
ZIK do re .b re mi .b mi fa sol .b sol la .b la si .b si do+
PLAY          ( joue une gamme chromatique)
```

Notes :

— fa .b donnerait un mi, et do+ .b un si.

— Aucun test de validité n'est effectué.

Evitez d'écrire "do .b", vous obtiendriez un silence...

Pour obtenir un si de l'octave inférieure, changez d'octave.

> MUS --- adr

Mot de bas niveau, donnant l'adresse du prochain octet disponible dans la table en cours de définition.

```
> MUS 1-        ( donne l'adresse de la dernière note compilée)
```

Attention, les paramètres occupent deux ou trois octets dans la table...

· MUS + n ---

Incrémente de *n* l'adresse du prochain octet disponible pour la table en cours. En fait, > MUS + est le mot système utilisé pour tenir à jour la longueur d'une table musicale au fur et à mesure de la compilation des notes et des paramètres musicaux. Voir MUSTAB pour la structure d'une table musicale.

DUR n ---

Fixe la durée des prochaines notes jouées, compte tenu du tempo. Les valeurs possibles de n vont de 1 à 96. La valeur à la mise sous tension est 24, ce qui avec un tempo moyen (5) correspond à une note noire.

Les durées sont proportionnelles à la valeur de n, de sorte que :

n = 6 pour une double croche.	n = 12 pour une croche.
n = 24 pour une noire.	n = 36 pour une noire pointée.
n = 48 pour une blanche.	n = 72 pour une blanche pointée.
n = 96 pour une ronde.	etc.

INITMUS ---

Met à zéro la table musicale actuellement sélectionnée, sans supprimer le mot du dictionnaire. La taille déclarée pour la table est inchangée, et vous pouvez recommencer à écrire votre partition à partir du début.

```
38 MUSTAB HAUT    38 MUSTAB BAS
: CREE-HAUT    HAUT 5 Ø DO I 1+ oct do mi sol do LOOP 4 oct ;
: CREE-BAS     BAS 5 Ø DO 5 I - oct do+ sol mi do LOOP 4 oct ;
CREE-HAUT  CREE-BAS  HAUT  PLAY
```

Nous avons confondu do avec do+ (avant le LOOP de CREE-HAUT). Si maintenant vous voulez réécrire la table HAUT, nous pouvons faire un FORGET HAUT et tout recommencer. Cette solution pourra être gênante si nous avons écrit beaucoup de définitions depuis... Si par ailleurs nous nous contentons de réécrire le mot CREE-HAUT et de l'exécuter, les notes seront compilées sans vérification à la fin de la table existante... Dans notre exemple, c'est la table BAS

qui sera affectée, le nom BAS disparaîtra du dictionnaire, etc. La bonne méthode dans ce cas est :

```
: RECREE-HAUT    HAUT    INITMUS
                 5 0 DO I 1+ oct do mi sol do+ LOOP 4 oct ;
RECREE-HAUT
```

MUSBUF --- adr

Donne l'adresse du troisième octet de la table musicale en cours d'utilisation.
Le nombre (16 bits) à cette adresse est la longueur actuellement occupée par la partition. Voir MUSTAB.

```
MUSBUF  2- ?        ( donne la longueur réservée pour la table )
MUSBUF  ?           ( donne la taille actuelle de la table )
                    ( sans tenir compte des quatre premiers octets )
MUSBUF  2+  C@ .    ( donne le code de la première note de la table )
                    ( ou du premier paramètre musical... )
```

MUSTAB n ---

Mot de définition utilisé sous la forme n MUSTAB XXX pour créer une table musicale pouvant contenir n octets en notes et paramètres musicaux.
Lors de l'exécution du mot XXX, la table XXX sera positionnée comme table courante. Les mots permettant de construire une table (mots en minuscules) concerneront alors la table XXX, ainsi que le mot PLAY et les mots système > MUS, MUSBUF, et > MUS+.
Soyez prudents lors de la création de la table. Conformément à l'esprit général du langage, aucune vérification ne sera faite lors de la compilation des notes. Par conséquent, si la place réservée était insuffisante, votre table empiètera sur la suite du dictionnaire (ou l'inverse, suivant l'ordre des opérations).

Au point de vue système, la structure d'une table musicale est la suivante:

	_____	Début de la table en mémoire
(1)	Longueur réservée (n ci-dessus), en deux octets	
	_____	Adresse donnée par MUSBUF
(2)	Longueur actuelle de la zone (3), en deux octets -	
	_____	
(3)	Codes des notes et des paramètres Chaque note occupe un octet (son code, de 48 à 61). Les altérations n'occupent aucune place (modifient la derniére note entrée). Les paramètres musicaux occupent chacun trois octets (sauf pitch, qui en occupe deux).	n octets
	_____	Adresse donnée par > MUS
	...	
	_____	Fin de la table

Il est facile, en consultant les codes des notes et l'effet des mots dur, temp, oct et pitch (voir ces mots) sur les tables musicales, de fabriquer un décompilateur de tables musicales.

NOTE n ---

Joue effectivement la note codée par le nombre *n*. *n* doit être un nombre entre 48 et 61. 48 est le code du silence (voir pause), 49 le code de do (voir do), 5Ø celui de do-dièse, 51 celui de re, et ainsi de suite en augmentant d'un demi-ton à la fois.
NOTE tient compte des paramètres musicaux en cours (voir DUR, TEMP, OCT, PITCH).

```
Musique au clavier : VARIABLE OCTAVE VARIABLE DUREE
: OCT+ (n ---) OCTAVE @ + 5 MIN 1 MAX DUP OCT OCTAVE ! ;
: DUR+ (n ---) DUREE @ SWAP 0< IF 2/ ELSE 2* THEN
               3 MAX 98 MIN DUP DUR DUREE ! ;
  : CLAV-ZIK 4 OCTAVE ! 24 DUREE !
     BEGIN KEY
       CASE
         " Q"   ASC OF 49 NOTE ENDOF  ( Q -- > do)
         " S"   ASC OF 51 NOTE ENDOF  ( S -- > re)
         " D"   ASC OF 53 NOTE ENDOF  ( D -- > mi)
                ( etc.)
         " Z"   ASC OF 50 NOTE ENDOF  ( Z -- > do-dièse)
         " E"   ASC OF 52 NOTE ENDOF  ( E -- > mi-bemol)
                ( etc.)
         10 OF -1 OCT+ ENDOF   ( Flèche vers le bas -- > + 1 octave)
         11 OF  1 OCT+ ENDOF   ( Flèche vers le haut-- > -1 octave)
          8 OF -1 DUR+ ENDOF   ( Flèche à gauche -- > diminue la durée)
          9 OF  1 DUR+ ENDOF   ( Flèche à droite -- > augmente la durée)
       ENDCASE PTRIG UNTIL ;
```

Ce mot, qui pourrait être écrit de manière plus condensée (par exemple en se servant d'une table des hauteurs des notes), donne par ailleurs un exemple intéressant de l'utilisation de CASE.

OCT n ---

Fixe l'octave des prochaines notes jouées. Les valeurs possibles de *n* vont de 1 (octave la plus grave) à 5 (la plus aiguë), et la valeur par défaut est 4.
Le do+ (ou 61 NOTE) d'une octave est identique au do (ou 49 NOTE) de l'octave suivante.

```
BELL      ( joue en fait un accord do-mi-sol-do+)
2 OCT BELL    5 OCT BELL    4 OCT BELL
```

Ou encore, 3 OCT 61 NOTE est identique à 4 OCT 49 NOTE

PITCH n ---

Agit d'une certaine manière sur l'enveloppe des notes jouées. Les valeurs théoriquement possibles pour n vont de 0 à 255, et la valeur par défaut est 0.
En pratique, n=0 donne un son genre « piano », et lorsque n aug-

mente le son se colore et s'amortit de plus en plus vite. Pour n = 2, on obtient un son genre « orgue ». Les valeurs supérieures à 10 ne sont guère intéressantes que pour certains effets spéciaux.

```
10 MUSTAB GAMME        GAMME do re mi fa sol la si do+
: TEST-PITCH 20 0 DO CR I . ." pitch"
                        I PITCH PLAY KEY DROP LOOP ;
GAMME TEST-PITCH...
```

PLAY ---

Joue la table musicale actuellement sélectionnée.

```
8 MUSTAB CLASSIK    CLASSIK do re mi fa sol la si do+
8 MUSTAB BLEUE      BLEUE  do re mi .b fa sol la si .b do+
PLAY                joue la gamme bleue ( la dernière positionnée )
CLASSIK PLAY        joue la gamme classique.
BLEUE PLAY          rejoue la gamme bleue...
```

TEMP n ---

Fixe le tempo pour les prochaines notes à jouer. Les valeurs possibles de *n* vont de 1 à 255, et la valeur par défaut est 5. En pratique, il est rare d'utiliser un tempo supérieur à 20...
L'intérêt de ce paramètre est que les durées relatives des notes (fixées par DUR) sont respectées, et qu'une même table peut être jouée à des « vitesses » différentes.

```
20 MUSTAB TEST-TEMP
TEST-TEMP 72 dur do 48 dur mi 36 dur sol 24 dur do+
```

Essayez : 2 TEMP PLAY puis 5 TEMP PLAY, puis 20 TEMP PLAY.

do ---

Compile le code de la note do (49) dans la table musicale en cours. La note ne sera effectivement jouée que lorsque la table sera exécutée, par le mot PLAY.
L'effet sera alors le même que si on tapait « 49 NOTE ». En particulier, la note sera jouée en tenant compte des valeurs des paramètres de durée (voir dur et DUR), de tempo (voir temp et TEMP), d'octave (voir oct et OCT), et d'enveloppe (voir pitch et PITCH).

do+ ---

Compile le code du do aigu de l'octave en cours (61) dans la table musicale sélectionnée. Voir do.

dur **n** ---

Compile dans la table musicale en cours une instruction de durée qui, à l'exécution de la table par PLAY, aura le même effet que « n DUR » en mode direct.
Cette compilation occupe trois octets dans la table. Le premier vaut 33 (code de la durée en décimal), et est suivi par la valeur *n* (en deux octets).

fa ---

Compile le code du fa (54) dans la table musicale en cours. Voir do.

la ---

Compile le code du la (58) dans la table musicale en cours. Voir do.

mi ---

Compile le code du mi (53) dans la table musicale en cours. Voir do.

oct **n** ---

Compile dans la table musicale en cours une instruction de définition de l'octave qui, à l'exécution de la table par PLAY, aura le même effet que « n OCT » en mode direct.
Cette compilation occupe trois octets dans la table. Le premier vaut 35 (code de octave), et est suivi par la valeur *n* (en deux octets).

pause ---

Compile dans la table musicale en cours le code du silence (48). A l'exécution de la table par PLAY, l'effet sera un silence dont la durée sera déterminée par la valeur en cours de la durée.
Le mot pause n'a donc pas la signification de la pause musicale, la traduction française serait le "silence".

```
96 dur pause    donne effectivement une pause.
48 dur pause    donne une demi-pause.
24 dur pause    donne un soupir.
12 dur pause    donne un demi-soupir.
```

pitch n ---

Compile dans la table musicale en cours une instruction de définition de l'enveloppe qui, à l'exécution de la table par PLAY, aura le même effet que « n PITCH » en mode direct.
Cette compilation occupe deux octets dans la table. Le premier vaut 32 (code du pitch), et est suivi par la valeur n (en 1 octet).

re ---

Compile le code du re (51) dans la table musicale en cours. Voir do.

si ---

Compile le code du si (6Ø) dans la table musicale en cours. Voir do.

sol ---

Compile le code du sol (56) dans la table musicale en cours. Voir do.

temp n ---

Compile dans la table musicale en cours une instruction de définition du tempo qui, à l'exécution de la table par PLAY, aura le même effet que « n TEMP » en mode direct.
Cette compilation occupe trois octets dans la table. Le premier vaut 34 (code du tempo), et est suivi par la valeur n (en deux octets).

Chapitre 8

Opérations sur la mémoire

Certains de ces mots (par exemple écrire, lire ou modifier un nombre à une adresse) sont certainement parmi les plus utilisés dans les applications écrites en FORTH.
D'autres agissent sur des portions de mémoire, et peuvent être à ce titre utilisés pour manipuler les chaînes de caractères, à cela près qu'ils n'agiront pas sur les longueurs des chaînes (attention !!!).

Nous avons placé dans ce chapitre tous les mots qui concernent la recherche dans le dictionnaire, mais qui ne concernent pas directement la compilation. Cela est justifié par le coté « ouvert » du langage. Il est relativement facile et très instructif d'étudier la structure du dictionnaire, et même de modifier des mots en RAM.

Les descriptions de ce chapitre supposent que vous connaissez la structure des mots du dictionnaire, et les termes qui s'y réfèrent (comme cf, lf, pf, cfa, etc.). Vous trouverez une explication sommaire des principaux termes employés dans le lexique (annexe G), et des précisions sur la structure du dictionnaire à l'annexe C.

! n adr. ---

Range le nombre n (nombre entier 16 bits) à l'adresse adr.
Utilisable directement (lorsqu'on connait l'adresse), mais le plus souvent adr est laissé sur la pile par l'appel d'une variable ou d'un tableau (ou éventuellement d'une constante ou d'une table, si on utilise des pointeurs).

```
VARIABLE  NBRE      ( NBRE laisse sur la pile l'adresse contenant sa valeur )
20  NBRE  !         ( Range 20 à l'adresse laissée par le mot NBRE )
                    ( La valeur de la variable NBRE sera donc 20 )
```

Voir aussi C! et D! (analogues, pour nombres n en 8 ou 32 bits), et @ et ? (pour relire les valeurs).

' --- adr (en mode interactif)
--- (en mode compilation)

Mot immédiat, utilisé sous la forme ' XX pour obtenir l'adresse du champ paramètre (pfa) du mot XX (ou un message d'erreur si XX n'est pas au dictionnaire). XX est le mot qui suit ' dans le flot d'entrée.
En mode interactif, l'adresse adr est déposée sur la pile ; en mode compilation, elle est compilée en tant que « littéral » dans la définition en cours.
Exemple : modification d'une constante.
La valeur d'une constante est rangée dans les deux premiers octets de son pf.
D'où :

```
365 CONSTANT JOURS/AN    JOURS/AN .  365  OK
```

Pour une année bissextile :

```
JOURS/AN  1+!            JOURS/AN .  366  OK
```

Si on utilise ' dans une définition, il sera exécuté lors de la compilation, laissant sur la pile l'adresse du mot suivant de la définition. Pour que ' agisse lors de l'exécution, il faut forcer sa compilation (ou utiliser FIND).

```
: DOUBLE [COMPILE] ' CFA DUP EXECUTE EXECUTE ;
DOUBLE BELL ( donnera deux coups de sonnette)
DOUBLE  XX ( ne fonctionnera bien que si XX ne modifie pas l'état de la pile...)
```

(FIND) **adr1 adr2 --- adr3 n1 1**

ou **--- 0**

Mot système, utilisé par –FIND pour la recherche d'un mot dans le dictionnaire.
Recherche à partir du mot dont le nfa est adr2 le mot dont le nf est égal à la chaîne rangée en adr1 (le premier octet étant l'octet de longueur). Le résultat est 0 si la recherche est infructueuse. Sinon, adr3 est le pfa du mot trouvé, et n1 l'octet de longueur du nf (et une valeur de vérité à 1 pour indiquer que la recherche a été positive).

+! **n adr ---**

Ajoute n au nombre entier 16 bits rangé à l'adresse adr (addition en mémoire).
L'effet de « n adr +! » est identique à celui de la séquence FORTH « adr @ n + adr ! », mais +! est plus rapide (écrit en langage machine).
Comme pour le mot !, adr est le plus souvent laissé sur la pile par l'appel d'une variable.
Exemple : VARIABLE AVOIR
Recette 200 F : 200 AVOIR +!
Dépense 130 F : -130 AVOIR +! ou 130 NEGATE AVOIR +!

–FIND **--- adr n 1**
ou --- 0

Utilisé sous la forme –FIND XX (XX est le mot qui suit –FIND dans le flot d'entrée) pour chercher XX dans le dictionnaire. La recherche s'effectue dans le vocabulaire CONTEXT.
Si XX est bien un mot du dictionnaire, –FIND laisse sur la pile son pfa (adr), l'octet de longueur de son nf (n), et une valeur de vérité « vrai ». Si XX ne figure pas au dictionnaire, –FIND laisse seulement une valeur de vérité « faux ».
–FIND est plutôt un mot système, utilisé par FIND.

1 + ! adr ---

Incrémente (augmente de 1) le nombre situé à l'adresse adr.
adr est le plus souvent laissé sur la pile par l'appel d'une variable.

```
VARIABLE COMPTEUR    COMPTEUR ? Ø OK
COMPTEUR 1+!         COMPTEUR ? 1 OK
COMPTEUR 1+!         COMPTEUR ? 2 OK, etc.
```

1 – ! adr ---

Décrémente (diminue de 1) le nombre situé à l'adresse adr.
adr est le plus souvent laissé sur la pile par l'appel d'une variable.
Voir 1 + !.

```
                  COMPTEUR ? 2 OK
COMPTEUR 1-!      COMPTEUR ? 1 OK
```

< CMOVE adr1 adr2 n ---

Copie la zone de *n* octets commençant à l'adresse adr1 (de adr1 à adr1 + n) sur la zone de n octets qui commence à l'adresse adr2, en commençant par la fin (d'abord (adr1 + n – 1) sur (adr2 + n – 1), puis (adr1 + n – 2) sur (adr2 + n – 2), ..., et en dernier, (adr1) sur (adr2). Si les zones de départ et d'arrivée ne se chevauchent pas, cette action est équivalente à celle de CMOVE. Mais si adr2 est dans la zone de départ (entre adr1 et adr1 + n – 1), on ne peut utiliser CMOVE (l'octet provenant de adr1 effacerait le contenu de adr2 avant que celui-ci ne soit copié...). D'une manière générale, il est conseillé d'utiliser CMOVE lorsque adr2 < adr1, et < CMOVE lorsque adr2 > adr1.
Exemple : méthode rapide pour décaler les éléments d'un tableau. Supposons par exemple que le tableau TAB, prévu pour contenir au maximum 5Ø nombres 16 bits, contienne actuellement 2Ø nombres (numéros Ø à 19), et qu'on veuille insérer un élément en sixième position (numéro 5). Il convient donc, avant d'affecter l'élément numéro 5, de décaler les quinze nombres de la fin du tableau (de 5 à 19 sur les numéros 6 à 2Ø)...
On peut écrire :

```
5  TAB                ( adresse de début de zone à transférer)
6  TAB                ( début de la zone d'arrivée)
15 2* <CMOVE          ( déplace 15 nombres de chacun 2 octets)
```

Cette méthode est nettement plus rapide que celle consistant à décaler les nombres un à la fois, par exemple en utilisant DO...LOOP.

? adr ---

Affiche le nombre 16 bits rangé à l'adresse adr.
L'effet est le même que celui de la séquence "@ ."...
Ce mot est utilisé la plupart du temps pour consulter la valeur d'une variable ou d'un élément d'un tableau de nombres 16 bits.

```
VARIABLE XX          XX ? 0 OK
34 XX ! 23 XX +!  XX ? 57 OK
```

?FIND --- adr

Laisse sur la pile le cfa du mot qui suit dans le flot d'entrée, et un message d'erreur si ce mot ne fait pas partie du dictionnaire.
Exemple : création d'une table de commandes...

```
TABLE COMMANDE
?FIND MOT2 , ?FIND MOT1 , ?FIND MOT2 , ?FIND MOT3 , etc.
```

Par la suite :

```
2 COMMANDE EXECUTE exécutera MOT2.
```

?FIND est un des mots de haut niveau (voir aussi FIND et ') utilisables dans vos applications pour trouver le cfa d'un mot, lorsque vous voulez qu'en cas d'erreur l'exécution se termine par un message d'erreur.
Exemple : le mot cherché a-t-il été défini par :...; ???

```
: ?: ?FIND @ ' HEX CFA = ( HEX est défini par :...; )
        IF ." est " (Son cf contient donc l'adresse d'exécution)
        ELSE ." n'est pas " THEN ( de ":". Voir annexe G)
        ." un mot défini par :...;"
         ;
```

Ce qui donne :

```
?: VLIST      est un mot défini par :...; OK
?: WHITE      n'est pas un mot défini par :...; OK ( constante )
?: #$]        UK ERROR OK       ( c'est du chinois !!!)
```

adr --- n

Dépose sur la pile le nombre rangé à l'adresse adr. Un des mots les plus utilisés du FORTH. La plupart du temps, adr est laissé sur la pile par l'appel d'une variable ou d'un tableau.

```
VARIABLE XX  34 XX !    VARIABLE YY    23 YY !
XX @ YY @ + .  57  OK
```

Les valeurs de XX et YY sont restées inchangées :

```
XX ? YY ? 34 23  OK
```

BANK n ---

Sélectionne la banque de 16 K de numéro n, si vous disposez d'une extension mémoire de plus de 16 K. Vous pouvez alors utiliser l'une des banques pour ranger des données en mémoire vive.
n doit être compris entre Ø et 5 (suivant la taille de votre système). La banque par défaut est la banque Ø.

```
: B1@  ( adr --- n )  1  BANK  C@  Ø BANK ;
```

pour lire l'octet situé à l'adresse adr dans la banque 1 (et revenir en banque Ø...).

BLANKS adr n ---

Met à blanc (en la remplissant de caractères « espace ») la zone de n caractères commençant à l'adresse adr.
Exemple : mise à blanc du block 12...

```
12  BLOCK  B/BUF  BLANKS
```

Ce mot peut être utilisé pour remplir d'espaces une chaîne de caractères, mais il n'agit pas sur sa longueur. Il est donc préférable d'utiliser dans ce cas le mot $!. Toutefois, lorsqu'on veut par exemple des affichages cadrés dans un champ de longueur fixe, on peut écrire :

```
1Ø STRING S     1Ø S DROP 1 - C!   ( fixe autoritairement la longueur à 1Ø)
S BLANKS        ( remplit S avec 1Ø caractères "espace")
```

Après avoir utilisé $!, il faudra refixer la longueur de S à 1Ø...
Pour éviter cette contrainte, on peut utiliser CMOVE au lieu de $!, de manière à ne pas agir sur l'octet de longueur de S... La chaîne S doit alors avoir été initialisée à sa longueur maximum, comme ci-dessus.

```
: INIT$ ( str - ) DDUP MLEN ROT DDUP 1-c4- C!
                  SWAP BLANKS DROP ;  ( Initialise str )
: LEFT$! ( str1 str2 --- )  DDUP BLANKS ROT MIN CMOVE ;
```

On pourra alors utiliser par exemple : `11 STRING S S INIT$`

```
" Coucou" S LEFT$! S TYPE Coucou          OK      ( longueur 11 )
" Salut" S LEFT$!  S TYPE Salut           OK      ( Itou !... )
" Je suis fourbu "  S LEFT$! S TYPE  Je suis fou OK   ( tronqué à 11 )
```

C! n adr ---

Range l'octet de poids faible de n à l'adresse adr.
Ce mot sert surtout à gérer des tableaux d'octets ou de caractères (en liaison avec le mot C@).
Il peut aussi être utilisé pour agir directement sur la mémoire, octet par octet (modifier une chaîne de caractères, écrire directement à l'écran, ...).

```
10 CARRAY TABLO   12 3 TABLO C!
```

pour ranger le nombre 12 en numéro 3 (quatrième valeur) dans TABLO.
Voir aussi C@ pour l'opération contraire.

C@ adr --- n

Lit l'octet situé à l'adresse adr, et le dépose sur la pile comme nombre 16 bits (l'octet de poids fort de n est à Ø).

```
10 CARRAY TABLO   12 3 TABLO C!
```

Pour relire le tableau : `3 TABLO C@ . 12 OK`

CFA adr1 --- adr2

adr1 étant le pfa d'un mot du dictionnaire, adr2 est son cfa.
Voir l'annexe G pour la structure des mots du dictionnaire.

CMOVE adr1 adr2 n ---

Comme < CMOVE (voir ce mot), CMOVE sert à déplacer la zone de mémoire de n octets commençant à l'adresse adr1 sur la zone de n octets commençant à l'adresse adr2. Mais l'opération de copie s'effectue dans l'autre sens.
CMOVE copie d'abord l'octet (adr1) sur (adr2), puis (adr1 + 1) sur (adr2 + 1), et ainsi de suite, pour terminer par la copie de (adr1 + n – c4-1) sur (adr2 + n – c4-1). Voir < CMOVE pour les conseils d'utilisation.
On peut reprendre l'exemple de < CMOVE, mais pour déplacer des nombres il est plus simple de se servir de MOVE. CMOVE sert plutôt à décaler des zones de mémoire... Par exemple, recopier le block 2 dans le block 34 :

```
2 BLOCK 34 BLOCK B/BUF CMOVE ( ou 1Ø24 CMOVE )
```

Voir aussi le dernier exemple du mot BLANKS (cadrage à gauche d'une chaîne de caractères)

CONTEXT --- adr

Variable permettant de sélectionner le vocabulaire « contexte », c'est-à-dire celui dans lequel l'interpréteur commence par chercher le mot à exécuter ou à compiler.
A l'adresse adr, on trouve le nfa du dernier nom du vocabulaire. CONTEXT est utilisé par FIND, ', et plus généralement par tous les mots qui font des recherches dans le dictionnaire.

EDITOR	positionne le vocabulaire de l'éditeur comme contexte.
VLIST	montre les mots de l'éditeur... puis ceux du FORTH.

D! d adr ---

Range le nombre 32 bits d à l'adresse adr. Ce mot est à mettre en parallèle avec ! (16 bits) et C! (8 bits).
adr est le plus souvent laissé sur la pile par l'appel d'une variable sur 32 bits (voir le mot DVARIABLE).

```
DVARIABLE XX 123456789. XX D!
```

Ou encore : INPUT XX D!
Voir le mot D@ pour l'opération inverse.

D@ adr --- d

Lit le nombre 32 bits situé à l'adresse adr, et le dépose sur la pile. Les 16 bits de poids faible sont sur le sommet de la pile.
Ce mot s'utilise souvent en liaison avec D!, pour manipuler des variables 32 bits (voir D! et DVARIABLE).

```
DVARIABLE XX INPUT XX D! (Tapez un nombre, puis la touche ENTREE)
XX  D@  D. ( Pour relire le nombre que vous avez tapé)
```

DP --- adr

Variable système contenant l'adresse du premier octet libre après la fin du dictionnaire.
HERE est défini par : `: HERE DP @ ;`

DUMP adr n ---

Affiche le contenu des n octets à partir de l'adresse adr. Le défilement peut être interrompu dans les mêmes conditions que celui du mot VLIST (toute frappe de touche arrête le défilement. Il reprend par l'appui de n'importe quelle touche autre que STOP, et s'arrête définitivement si on appuie sur STOP).
L'affichage se fait par lignes de huit octets (si n vaut 10, DUMP affichera 16 octets), et chaque ligne comporte :
— adresse du premier octet affiché sur la ligne (suivi de ":").
— valeurs hexadécimales des huit octets à partir de cette adresse.
— affichage ASCII de ces huit octets. Les caractères de contrôle (codes 0 à 31) sont représentés par des points.

```
 0  20  DUMP     ( affiche 24 octets - 3 paquets de 8 octets)
   0: 20 46 4F 52 54 48 2D 37    FORTH-7
   8: 39 20 20 53 2E 45 2E 45   9  S.E.F
  10: 2E 49 2E 20 31 2E 30  4  .I.  1.0.
```

L'octet 8 de la mémoire contient le nombre 39 (affichage hexadécimal, qui correspond à 57 en décimal), qui est le code ASCII du chiffre 9. L'octet 11 (B en hexadécimal) contient le nombre 53, qui est le code ASCII de la lettre "S". L'octet 16 (10 en hexadécimal) contient le nombre 2E, code du caractère ".". L'octet 24 (le dernier) contient le nombre 4, qui correspond au caractère "CNT-D" (caractère de contrôle) et est donc affiché comme un point.

Le mot DUMP est très intéressant pour étudier la structure interne de FORTH, mais aussi lors de la mise au point des applications, pour vérifier le contenu des mots. A cet effet, les mots souvent utiles sont ' ou FIND (pour donner les adresses des mots à « dumper »), NFA (pour trouver le début de la zone à examiner et pour utiliser ID.), ainsi que ID. (qui affiche un nom lorsqu'on lui donne un nfa). Il est ainsi possible de « décompiler » n'importe quel mot écrit en FORTH.

```
FIND BLANKS 2+ NFA 20 DUMP
C28: 86 42 4C 41 4E 4B D3 C                éBLANK..
C30:  7 11 AF  B F3  8 AA 9                ........
C38:  1 .....                              La suite appartient au
                                           mot ERASE.
```

86:	Longueur du mot, et mot non immédiat...) nf
42 4C 41 4E 4B D3:	Nom du mot) nf
C07:	nfa du mot précécent) lf
	Ici: HEX C07 ID . SPACES OK (VLIST pour vérifier)
11AF:	Adresse d'exécution de ":") cf
BF3, 8AA:	cfa des mots composant BLANKS...) pf
	BF3 2+ NFA ID. BL OK 8AA 2+ NFA ID. FILL OK
901:	Adresse d'exécution de ";") marque la fin du pf

Autrement dit, le mot BLANKS est défini de la manière suivante :

```
: BLANKS BL FILL ;
```

EDITOR ---

Sélectionne le vocabulaire de l'éditeur comme contexte (voir les mots CONTEXT et VOCABULARY). Voir en annexe les commandes disponibles sous l'éditeur.

ERASE adr n ---

Met à zéro (en remplissant avec le caractère de code 0) la zone de n octets à partir de l'adresse adr. L'effet de ce mot est très voisin de celui du mot BLANKS, la seule différence est que la zone considérée est remplie avec le caractère de code 0 au lieu du caractère « espace » (code 32).
Si vous utilisez ERASE pour mettre à zéro un block, il faut savoir que ce block ne peut plus être édité (à moins de le remettre à blanc),

car le code Ø sert en FORTH à marquer la fin du flot d'entrée. Par conséquent, le mot LOAD s'arrêtera dans l'interprétation au premier Ø rencontré.
Réinitialisation d'un tableau de 3Ø nombres... (Le nombre Ø est codé en 16 bits par deux octets à Ø.)

```
Ø TABLO 3Ø 2* ERASE
```

FENCE --- adr

Variable contenant l'adresse en dessous de laquelle le dictionnaire est protégé. Les mots dont le nfa est inférieur au contenu de FENCE ne peuvent être oubliés par FORGET.

`HERE FENCE !`	protège tout le dictionnaire.
`' XX NFA FENCE !`	Il est maintenant impossible d'oublier les mots définis antérieurement à XX (sauf si vous modifiez FENCE...).

Le dictionnaire chargé depuis la cassette ou la disquette sous forme compilée (par RUN, ou en sélectionnant "UN PROGRAMME ENREGISTRE" au démarrage du système) est toujours protégé. Pour utiliser FORGET (si vous voulez modifier une application < mot >), faites :

```
Ø FENCE ! FORGET <mot>
```

Même lorsque FENCE a la valeur Ø, il est bien entendu impossible d'oublier les mots de la cartouche-ROM (voir en annexe l'implantation mémoire du langage).

FILL adr n b ---

Remplit la zone de *n* octets à partir de l'adresse adr avec le caractère *b*. *b* doit être un code ASCII, c'est-à-dire inférieur ou égal à 255.

```
2 BLOCK 4ØØ 65 FILL
```

remplit les 1Ø premières lignes du block 2 avec des « A ».
Pour voir le résultat, faites "2 EDIT" ou "2 LIST"...
ERASE est défini par : `ERASE Ø FILL ;`
BLANKS est défini par : `BLANKS BL FILL ;`
La raison d'être de ces mots est qu'ils sont utilisés à plusieurs reprises par le système, et que le code FORTH est plus court en les définissant.

FIND --- adr

Mot utilisé pour la recherche dans le dictionnaire. FIND laisse sur la pile l'adresse de compilation (le cfa) du mot qui le suit dans le flot d'entrée. Si ce mot ne figure pas dans le dictionnaire, le résultat de FIND est Ø.
Exemple : remplir une table de commande (voir ?FIND . La différence est que FIND ne provoque pas de message d'erreur si le mot cherché n'est pas trouvé, mais laisse simplement la valeur Ø sur la pile)

FIRST --- adr

Donne l'adresse du début du premier buffer disponible. Voir en annexe l'implantation mémoire du FORTH.

```
HEX FIRST U.  77F8 OK
```

(TO7 sans extension mémoire, avec deux buffers.)
Taille de la zone disponible pour le dictionnaire:

```
FIRST PAD - DECIMAL 84 - . 4966 OK
```

(Un peu moins de 5 K.)

FORGET ---

FORGET XX supprime du dictionnaire le mot XX et tous les mots qui ont été définis postérieurement à XX. La recherche de XX se fait d'abord dans le vocabulaire sélectionné comme « vocabulaire courant » (voir CURRENT, chapitre 4), puis éventuellement dans le vocabulaire FORTH.
FORGET ne peut supprimer les mots dont le nfa est inférieur à la valeur contenue dans la variable FENCE (voir ce mot), ni les mots du FORTH de base (puisqu'ils sont en mémoire morte).
Il est souvent commode de commencer l'écriture d'une application par un mot « vide », un mot qui ne fasse rien. Par la suite, vous pourrez oublier toute votre application en oubliant ce mot.

```
: XX ;                    ( mot vide )
.....                     ( votre application )
FORGET XX                 ( oublie toute l'application )
```

Les premiers FORTH disposaient même d'un mot à cet effet (mot vide), le mot TASK...

FORTH ---

Sélectionne le vocabulaire FORTH comme contexte (voir CONTEXT). En fait, ce mot désélectionne le vocabulaire antérieurement positionné, puisque tout vocabulaire sélectionné (par exemple VOCAB) est considéré comme un « prolongement » du dictionnaire (en ce sens qu'une éventuelle recherche se ferait d'abord dans VOCAB, puis en cas d'insuccès dans FORTH).
FORTH est donc le vocabulaire minimum, le noyau sur lequel se greffent les autres vocabulaires.

```
EDITOR  VLIST ...
FORTH   VLIST ...          Les mots de l'éditeur ont disparu.
```

Dès que l'interpréteur rencontre une erreur, il repositionne autoritairement FORTH comme vocabulaire contexte.

```
EDITOR  VLIST ...
CCXCC                      Provoque un "UK ERROR"
VLIST                      Les mots de l'éditeur ne sont plus là.
```

HERE --- adr

Donne l'adresse qui suit la fin du dictionnaire. C'est à cette adresse que commencera la prochaine définition.

```
HERE  .   25567 OK          ( A la mise en route du système)
: MOT  BELL  RED  FRAME  ;  ( Un mot pour voir ...)
' MOT  NFA  .   25567 OK
```

ID. adr ---

Affiche le nom d'un mot à partir de son nfa.
Exemple : décompilation d'un mot.

```
' ABS  DUP  @  2+  NFA  ID.  DUP OK
2+  DUP  @  2+  NFA  ID.  +- OK
2+  DUP  @  2+  NFA  ID.  ;S OK
```

Autrement dit, le mot ABS est défini par: : ABS DUP +- ;
Dans cet exemple (qui ne peut décompiler que des mots définis par :..;), on lit l'une après l'autre les valeurs contenues dans le pf de ABS, qui sont les cfa des mots qui constituent ABS. La fin de la définition est repérée par le mot ";S" (voir annexe C, relative à la structure du dictionnaire).

LATEST --- adr

Donne le nfa du dernier mot défini dans le vocabulaire actuellement sélectionné comme courant (voir CURRENT, chapitre 4).

```
EDITOR DEFINITIONS LATEST ID. EDSCR OK
```

LATEST est surtout utilisé par le système, lors de la création de l'entête de nouveaux mots.

LFA adr1 --- adr2

Donne le lfa d'un mot à partir de son pfa. Le lf d'un mot contient le nfa du mot qui le précède dans le dictionnaire (voir annexe relative à la structure des mots du dictionnaire)

Exemple : parcours du dictionnaire.

```
' DUMP NFA DUP ID. DUMP OK
PFA LFA @ DUP ID. VLIST OK
PFA LFA @ DUP ID. 1-! OK etc.
```

LIMIT --- adr

Donne l'adresse qui suit la plus haute adresse utilisable par le système, suivant la configuration.

Exemple : avec un TO7

```
LIMIT HEX U. 8000 OK        ( sans extension mémoire)
LIMIT U. C000 OK            ( avec extension de 16 K)
```

Voir en annexe l'implantation mémoire du FORTH, en particulier des buffers.

En fait, LIMIT – 1 est l'adresse de fin du dernier buffer accessible.

MOVE adr1 adr2 n ---

Analogue à CMOVE. MOVE sert à déplacer la zone de mémoire de *n* nombres commençant à l'adresse adr1 sur la zone de *n* nombres commençant à l'adresse adr2.
CMOVE copie d'abord le nombre (adr1,adr1 + 1) sur (adr2,adr2 + 1), puis (adr1 + 2,adr1 + 3) sur (adr2 + 2,adr2 + 3), et ainsi de suite...
Exemple : énoncé complet dans < CMOVE. Nous voulons maintenant supprimer le huitième nombre du tableau TAB (il comporte toujours 2Ø nombres 16 bits). Il s'agit donc de décaler les éléments du tableau à partir du neuvième (numéro 8) jusqu'au dernier (numéro 19), sur les éléments numéros 7 à 18...

8 TAB	7 TAB	12	MOVE
Début de la zone de départ	Début de la zone d'arrivée	Nombre de nombres à décaler (19 - 8 + 1)	

NFA adr1 --- adr2

Donne le nfa d'un mot à partir de son pfa. Voir en annexe C la structure des mots du dictionnaire, ainsi que ID. et LFA.
Exemple : modification du nom d'un mot du dictionnaire (en mémoire vive)...

```
: MOT  BELL  ;          ( Mot pour essayer...)
'  MOT  NFA  2+         ( adresse du caractère "O" du nom "MOT")
65 SWAP C!              ( remplace O par A, de code 65)
VLIST ...               ( MOT est devenu MAT)
```

PFA adr1 --- adr2

Donne le pfa d'un mot à partir de son nfa. Voir en annexe C la structure des mots du dictionnaire, et LFA (parcours du dictionnaire).

TOGGLE adr c ---

Modifie l'octet situé à l'adresse adr en réalisant un « ou exclusif » avec l'octet c.
Utilisation fréquente : modifier un bit dans un octet (par exemple, dessiner en écrivant directement dans la mémoire écran, gérer une interface, etc..)
Exemple : description d'un système électrique comportant huit bascules (Ø à 7), en utilisant un octet comme tableau de booléens.

```
VARIABLE ETAT  ( seulement 8 bits seront utilisés...)  8 CTABLE NUM
1 C, 2 C, 4 C, 8 C, 16 C, 32 C, 64 C, 128 C,
: BASCULE  ( n --- )    ETAT SWAP 1- NUM TOGGLE ;
```

5 BASCULE modifiera le cinquième bit de ETAT (et pas les autres), simulant l'action « basculer le cinquième interrupteur »... Pour tester la position d'un interrupteur, on peut utiliser AND, et pour forcer l'ouverture ou la fermeture on utilisera AND ou OR.

Par exemple :
— Tester la troisième bascule :

```
ETAT @ 3 NUM AND Ø= NOT
```

donnera 1 si l'interrupteur 3 est ouvert (bit 4 à 1), et Ø s'il est fermé.
— Ouvrir l'interrupteur 4 (bit 5) :

```
ETAT @ 4 NUM OR ETAT !
```

TRAVERSE adr1 n --- adr2

Donne l'adresse d'une extrémité du nf d'un mot en fonction de l'autre. Si adr1 est le nfa et si n vaut 1, adr2 est l'adresse du dernier octet du nf. Si adr1 est l'adresse de fin du nf et si n vaut −1, adr2 est le nfa du mot.
Le mot NFA est défini par :

```
: NFA 5 - -1 TRAVERSE ;
```

("5−" donne l'adresse de fin du nf d'un mot à partir de son pfa.)

VLIST ---

Donne la liste des mots du dictionnaire, en commençant par le dernier nom du vocabulaire sélectionné comme contexte.
Le défilement peut être interrompu à n'importe quel moment par la frappe d'une touche quelconque. Il reprend dès qu'on frappe une touche autre que la touche STOP (si on appuie la touche STOP, il s'arrête définitivement.

WIDTH --- adr

Variable contenant le nombre de caractères qui seront pris en compte pour les noms des mots, dans les définitions ultérieures. Attention, la longueur mémorisée est la longueur du nom effectivement tapé !!! Il appartient au programmeur de respecter la longueur qu'il a annoncée...

```
2 WIDTH !
: ROUGE RED FRAME ; RO RO ? UK ERROR  ( La longueur du nom est 5)
```

Si vous tapez ROxxx, ou ROOOO, ou n'importe quel nom de cinq caractères, RO s'exécutera. Vous pouvez vérifier ce qui s'est passé en faisant un DUMP:

```
' ROUGE NFA 10 DUMP
```

Les premiers octets sont : 85 52 CF
85 signifie « longueur 5 » (voir en annexe C la structure d'un nom).
52 et CF sont les codes ASCII de R et de O.
Le dernier code a son bit 7 à 1 pour indiquer la fin du nom.
Si maintenant vous définissez un autre mot avec un nom de cinq caractères commençant par RO (par exemple : RONCE BELL ;), vous aurez un message « Not Unique ». FORTH ne fait plus de différence entre ROUGE et RONCE.
Par contre, un nom de longueur différente ne posera pas de problème. Exemple :

```
: ROTI RED PAPER COLOR INK BLACK COLOR BELL BELL ;
```

ROTI (ou ROUX) donne bien les deux coups de sonnette...
Les mots définis antérieurement ne suivent pas cette logique, puisque leurs noms sont écrits en entier. Par exemple, VLXXX ne fait pas un VLIST, il faut écrire VLIST en entier.
Il est donc à conseiller, si vous avez défini une longueur de deux caractères, de ne pas dépasser cette longueur dans les noms des mots que vous choisissez pour vos nouvelles définitions.

Chapitre 9

Entrées-sorties (sauf cassettes et disquettes)

Nous nous intéressons ici à tous les moyens de faire communiquer votre ordinateur avec son environnement (dont vous faites partie), excepté avec la "mémoire externe" (cassettes ou disquettes). Cet environnement comporte, en entrée (dans l'ordinateur) : le clavier, le crayon optique et les manettes de jeux. En sortie (de l'ordinateur) : l'écran et l'imprimante.

La structure de FORTH fait que contrairement à d'autres langages, il est nécessaire d'utiliser des mots pour lire (au clavier) ou écrire (à l'écran) qui sont spécifiques à chaque type de donnée (nombres 16 bits, 32 bits, caractères ou chaînes de caractères). Par ailleurs, cela est aussi un facteur de rapidité du langage, puisque les mots d'affichage n'ont pas à tester le type de donnée qu'ils manipulent.
En ce qui concerne l'affichage, les mots sont en revanche les mêmes pour l'écran et l'imprimante. Votre FORTH dispose de deux modes, le mode "CONSOLE" et le mode "PRINTER" (voir ces mots). Il suffit de positionner l'un ou l'autre (le mode "CONSOLE" est fixé par défaut) avant affichage. En fait, les mots de base sont EMIT et CR, et donnent lieu chacun à deux mots : (EMIT) et (CR) pour l'affichage-écran et (PEMIT) et (PCR) pour l'affichage à l'imprimante. Suivant le mode, EMIT et CR exécutent les mots adéquats, dont les cfa sont en positions 1 et Ø dans le tableau SYSVEC (voir

annexe D, relative aux mots systèmes). Il est possible d'utiliser (EMIT) au lieu de EMIT pour afficher à l'écran, mais ceci est à déconseiller. D'une part EMIT est standard (question de portabilité de vos applications), d'autre part vous perdrez en souplesse (possibilité de modifier le périphérique de sortie) plus que vous ne gagnerez en rapidité...

Toutes les données lues au clavier sont du type caractère, ou chaîne de caractères (lorsque vous tapez le nombre 12, vous tapez en fait les caractères 1 puis 2).

La longueur maximum d'une chaîne de caractères est 255 caractères. S'il est possible de taper 255 caractères en réponse à LINPUT$ (voir ce mot), les chaînes entrées au clavier en mode "définition de mots" doivent comporter 80 caractères au plus (taille du buffer d'entrée, voir le mot TIB). Lorsque vous tapez une définition (sans utiliser l'éditeur), il est donc important de taper la touche ENTREE avant le quatre-vingtième caractère (FORTH interprète alors les caractères tapés, vide le buffer d'entrée, et attend la suite). Si vous ne le faites pas, le risque est que le quatre-vingtième caractère soit au milieu d'un nom de mot, et que FORTH soit incapable d'interpréter votre texte...

Les nombres sont lus en tant que chaînes de caractères (suites de chiffres), et convertis par le système en nombres qui sont déposés sur la pile. L'affichage opère en sens inverse, convertissant des nombres en suites de chiffres.

Les nombres 16 bits sont saisis de manière habituelle (sans signe "+" pour les nombres positifs). Pour prévenir le système que le nombre tapé est à considérer comme nombre 32 bits, il suffit d'y incorporer un point (sauf réponse à INPUT ou LINPUT. Voir ci-dessous). Ce point n'a rien à voir avec un quelconque point décimal, encore que la position du point est sauvegardée dans la variable DPL. Vous pouvez vous servir de la valeur de DPL pour réaliser un affichage en virgule fixe.
Par contre, le nombre tapé en réponse à INPUT ou LINPUT sera toujours un nombre 32 bits. Ici, les chiffres ou caractères qui suivraient un éventuel point décimal seraient ignorés.

```
INPUT D. ( Tapez: 1200.765. La pile contient le nombre 1200., nombre 32-bits)
```

Pour ces conversions entre nombres double longueur et chaînes de caractères, FORTH se réfère à la valeur de la variable BASE. Par exemple, "A" est un symbole valable en base seize, mais pas en base dix.

La zone mémoire utilisée pour ces conversions numériques (voir "Conversions", au chapitre 5) est la zone située entre la fin du dictionnaire et le début de la zone réservée au rangement temporaire des chaînes de caractères (voir le mot PAD, dans la section "chaînes de caractères").

(.") ---

Mot système compilé dans la définition en cours par le mot ." (en mode compilation) pour réaliser l'affichage d'une chaîne de caractères définie comme littéral. L'action de (.") lors de l'exécution de la définition sera d'afficher la chaîne littérale sur le périphérique en cours. La chaîne elle-même est compilée immédiatement après le mot (."), sous la forme : octet de longueur suivi des caractères de la chaîne.

(CR) ---

Mot système qui réalise le passage à la ligne à l'écran. (CR) est appelé par CR en mode "CONSOLE". (Voir aussi (PCR), pour le mode "PRINTER").
A la mise sous tension, l'adresse de (CR) est en position Ø dans le tableau SYSVEC. Si vous écrivez une autre routine, il vous suffit de mettre son cfa dans SYSVEC pour qu'elle soit appelée à la place de (CR) à chaque fois que cela est nécessaire...

```
: CR-- (CR) .' ----------------------------------------" ;
                                    ( Affiche un passage à la ligne et 40 tirets...)
FIND CR-- Ø SYSVEC !
```

Dorénavant, les lignes seront séparées par une ligne de tirets.
Par exemple : 1 Ø SIZE VLIST ...

(EMIT) c ---

Mot système appelé par EMIT pour afficher le caractère de code ASCII c sur l'écran. Le caractère c n'est pas nécessairement un caractère affichable, il peut aussi traduire une action sur l'affichage. Par exemple, 12 (EMIT) efface l'écran.

En ce qui concerne les caractères affichables, (EMIT) affiche le jeu complet de caractères ASCII étendu, y compris les caractères graphiques que vous aurez définis. Les minuscules accentuées et autres caractères spéciaux correspondent à un code ASCII unique (pas à une séquence), ce qui permet aux mots traitant les chaînes de caractères de fonctionner correctement.

Les caractères sont affichés compte tenu de la couleur (voir COLOR), de la taille (voir SIZE). Le backspace (code 8) n'est pas doublé pour les caractères en double largeur, mais il est facile de remédier à cette lacune :

```
: (EMIT DUP 8 = IF DUP (EMIT) THEN (EMIT) ;
FIND (EMIT 1 SYSVEC !
```

Puis : 1 1 SIZE (ou 0 1 SIZE ...)
La touche EFF fonctionne normalement

(GPRT) ---

Ce mot est appelé par le mot SCRPRT, et réalise une copie de l'écran graphique (point par point) sur l'imprimante thermique. Si vous disposez d'une imprimante graphique d'un autre modèle, vous pouvez écrire un mot analogue, et placer son cfa en position 11 du tableau SYSVEC. C'est lui qui sera appelé par SCRPRT.

(KEY) --- c

Mot système appelé par KEY pour réaliser la lecture d'un caractère au clavier Les caractères accessibles au clavier par des séquences de touches (par exemple les caractères accentués) sont convertis en un code unique (de 128 à 149).
Les caractères [et], qui sont utilisés par FORTH (voir par exemple les mots [et] au chapitre 4), s'obtiennent par CNT-A et CNT-Z (les codes ASCII 1 et 26 sont transformés respectivement en 91 et 93).

(PCR) ---

Mot système appelé par CR en mode "PRINTER". Envoie un retour chariot et un passage à la ligne à l'imprimante (carriage return + linefeed)..
Si votre imprimante fait un saut de ligne automatiquement, il vous faudra remplacer (PCR) par (PCR1), comme ci-dessous.

```
: (PCR1) 13 (PEMIT) ; FIND (PCR1) 10 SYSVEC !
```

(PEMIT) c ---

Mot système appelé par EMIT en mode "PRINTER". L'utilisation de (PEMIT) est analogue à celle de (EMIT) (position 9 du tableau SYSVEC), à cela près que l'affichage se fera sur l'imprimante, et que les caractères graphiques ne fonctionneront pas de la même manière.
- Les caractères de code ASCII inférieur à 127 ou supérieur à 149 sont envoyés tels quels.
- Les caractères de code ASCII compris entre 128 et 149 (voyelles accentuées) sont rectifiés de façon à écrire la voyelle non accentuée.
Si votre imprimante peut envoyer des caractères accentués, vous pouvez modifier (PEMIT) comme suit :
(Elle attend par exemple les codes $7B pour "é", $7D pour "è", $5C pour "ç"...)

```
HEX
: (PEMIT-F)    ( c ---)
    CASE 86 OF 7B ENDOF          ( Pour un "é", remplacer $86 par $7B)
         85 OF 7D ENDOF                          ( cas d'un "è")
         80 OF 5C ENDOF                          ( cas d'un "ç")
         ...                          ( autres caractères spéciaux ...)
         DUP                      ( autres cas : dupliquer pour ENDCASE)
    ENDCASE
    (PEMIT) ;              ( avant d'afficher le caractère comme d'habitude)
FIND (EMIT-F) 9 SYSVEC !
```

Votre imprimante affichera maintenant correctement les caractères spéciaux, à condition bien sûr qu'elle soit initialisée en caractères français.
En fait, (PEMIT) peut servir à envoyer un caractère sur le port de communication, et pas seulement à l'imprimante. Dans le cas d'une transmission série, il faudra initialiser l'opération d'écriture et la terminer en utilisant le mot (PIO) (voir chapitre 10).

(PKEY) **--- c**

Mot système qui réalise la lecture du port de communication, et dépose le caractère lu sur la pile. (PKEY) réalise donc en sortie ce que (PEMIT) réalise en entrée.
(PKEY) dépose toujours un caractère sur la pile même s'il rencontre une erreur de lecture. Dans ce cas, le code de l'erreur est dans la variable D-ERR (voir ce mot, chapitre 10). Avant de lire le premier caractère d'un ensemble de données, il faudra initialiser l'opération de lecture par 1 (PIO) (voir ce mot, au chapitre 10). Pour terminer la lecture, utilisez Ø (PIO).

. n ---

Affiche sur le périphérique en cours le nombre n (16 bits signé), en tenant compte de la BASE en cours.

```
54 HEX . FE OK
DECIMAL 254 2 BASE ! . 11111110 OK
```

Cet affichage est suivi d'un espace. Pour afficher des nombres sans séparation, utiliser .R
Voir les mots U. et U.R (16-bits non signés), ainsi que D., D.R, UD. et UD.R (affichage des nombres 32-bits).

." **---**

Utilisé sous la forme ." xxxxxx" pour réaliser l'affichage de la chaîne de caractères xxxxxx.
Le premier guillemet doit être suivi d'un espace.
Le guillemet final n'est pas un mot FORTH, et peut donc ne pas être précédé d'un espace. Il doit par contre être suivi d'un espace pour isoler le mot suivant. La chaîne à afficher ne peut pas comporter de guillemet. Par contre,elle peut contenir des caractères spéciaux (obtenus par les touches spéciales, par exemple les flèches ou la touche RAZ).
Le mot ." agit différemment suivant qu'on est en mode compilation ou en mode interactif.
En mode interactif, la chaîne xxxxxx s'affiche dès la validation par la touche ENTREE.
En mode compilation (dans une définition par :..;), le mot ." exé-

cute le mot (."), qui compile la chaîne xxxxxx comme littéral dans la définition en cours. L'affichage aura lieu lorsque la définition sera exécutée.

```
." Bonjour, tout le "  ." monde"                    ( Mode interactif)
: SALUT ." Bonjour tout le monde" BELL BELL BELL ;   ( Mode compilation)
```

.R n p ---

Affiche le nombre n justifié à droite dans une zone de p blancs, compte tenu de la base utilisée. Contrairement au point, cet affichage n'est pas suivi d'un blanc. Si la zone n'est pas assez large, .R se comporte comme le point (sauf affichage du blanc final).

```
DECIMAL 30000 HEX ." >> " 6 .R " << " >> 7530 << OK
CR 43 57 DDUP . . ." ------ " 0 .R 0 .R
57 43 -----5743 OK
```

Ce mot est souvent utilisé pour cadrer les affichages en tableaux.

```
: CARRES  ( n1 n2 --- Validité : -181 < n1 < n2 < 181)
        1+ SWAP DO CR I 5 .R I I * 5 .R LOOP ;
```

Comparez le résultat avec celui de :

```
: KARRES 1+ SWAP DO CR I I * . LOOP ;
```

Essayez 0 20 CARRES et 0 20 KARRES ...

>IN --- adr

Variable système donnant l'adresse relative au début du flot d'entrée du prochain caractère à traiter. Voir aussi IN?.

>OUT --- adr

Variable contenant le nombre de caractères affichés depuis le dernier passage à la ligne. Cette valeur est incrémentée par EMIT, et peut être utilisée pour le formatage de vos affichages. VLIST et l'éditeur comparent sa valeur à celle de la variable C/L (voir ce mot, au chapitre 11), de sorte que vous pouvez par exemple obtenir la liste des mots du dictionnaire en une seule colonne en tapant :

```
2 C/L ! VLIST                ( L'éditeur aussi n'utilisera que 2 colonnes...)
```

Ou à l'imprimante (en 80 colonnes) :

```
PRINTER 80 C/L ! VLIST CR 40 C/L ! CONSOLE
```

?STOP --- cond

Une commodité du FORTH, qui permet d'interrompre provisoirement un mot par la frappe d'une touche. L'ordinateur attend alors la frappe d'une autre touche, et laisse sur la pile la valeur de vérité répondant à la question : "A-t-on appuyé sur la touche STOP ?" cond vaudra 1 (VRAI) si on a appuyé sur la touche STOP, et Ø (FAUX) dans le cas contraire.

```
: BIPAGOGO (---) BEGIN ?STOP NOT WHILE BELL REPEAT ;
```

BIPAGOGO... Tapez une touche, ça s'arrête. Retapez une touche (pas la touche STOP), ça recommence... Recommencez, mais en tapant la touche STOP au deuxième coup : l'exécution s'arrête définitivement. VLIST, LIST, DUMP utilisent cette commodité.

?TERMINAL --- cond

Teste si une touche a été frappée, et laisse sur la pile la valeur de vérité cond. Contrairement à INKEY, la "valeur" de la touche n'est pas prise en compte, et le résultat de ?TERMINAL est uniquement qualitatif.

cond = 1 signifie "Oui, on a tapé une touche".
cond = Ø signifie "Non, on n'a pas tapé de touche".

L'utilisation la plus fréquente est comme test d'arrêt de boucle...

```
: GLOUSSE (---) BEGIN BELL ?TERMINAL UNTIL ;
```

BASE --- adr

Dépose sur la pile l'adresse d'une variable contenant la base actuellement utilisée pour la conversion des nombres en entrée ou en sortie. Voir chapitre 5.

BL --- n

Une constante, qui dépose sur la pile le code ASCII du caractère "espace" (32 en décimal).

Exemple : a-t-on tapé la touche < espace > ?

```
: TEST BEGIN KEY BL = NOT
       WHILE CR " Bof !... " REPEAT
       ." Ah, tout de même !!!" ;
```

Cette constante est très utilisée par le système (les espaces servent de séparateurs entre les mots FORTH) et par l'éditeur (les caractères des écrans sont tous des espaces, tant que vous ne tapez rien).

BS --- n

Une constante, qui dépose sur la pile le code ASCII du caractère "backspace" (c'est à dire 8), et qui, lorsqu'il est "affiché", a pour effet de faire reculer le curseur. Ce code est accessible au clavier par la touche "flèche à gauche" (et pas la touche "DEL").

```
." ABCDE" BS EMIT ." FG" ABCDFG OK
```

BUTTON n --- cond

Dépose sur la pile une valeur de vérité qui indique si on a appuyé sur le bouton de la manette de jeux numéro n (n ne peut valoir que Ø ou 1).

```
: TEST-BOUTONS ( --)
      BEGIN CR Ø BUTTON . 1 BUTTON . PTRIG
      UNTIL ;
  TEST-BOUTONS
```

Pour arrêter, appuyez le bouton à l'extrêmité du crayon optique...

CONSOLE ---

Sélectionne l'écran comme périphérique de sortie. L'affichage se fera dorénavant à l'écran... CONSOLE sert le plus souvent à arrêter l'action du mot PRINTER.

En fait, CONSOLE sélectionne comme routine d'affichage d'un caractère le mot dont le cfa est en position 1 dans le tableau SYSVEC, et comme routine de passage à la ligne le mot dont le cfa est en position Ø. Si vous n'avez pas modifié ces valeurs depuis la mise sous tension, les mots sélectionnés sont (EMIT) et (CR).

```
PRINTER VLIST CONSOLE
```

CR ---

Réalise un passage à la ligne de l'affichage. Pour cela, CR envoie un "Retour-chariot" (code ASCII 13) et un "Passage à la ligne" (code ASCII 10) au périphérique de sortie en service. CR exécute le mot (CR) en mode "CONSOLE", et le mot (PCR) en mode "PRINTER" (voir les mots CONSOLE et PRINTER).
En fait, CR appelle simplement le mot dont l'adresse est en position du tableau SYSVEC. Cette adresse peut être modifiée par les mots PRINTER ou CONSOLE, ou encore par vous, si par exemple vous disposez d'un périphérique spécial (voir exemple de (PCR)).

D. d ---

Affichage d'un nombre 32 bits signé (suivi d'un espace).

```
.33 D. 33 OK
```

Voir aussi D.R, UD. et UD.R

D.R d n ---

Analogue à .R, mais affiche un nombre 32 bits signé. (n est un nombre 16 bits).
Exemple : afficher la table des cubes de n1 à n2.

```
: CUBES    ( n1 n2 --- Validité : -1290 < n1 < n2 < 1290)
        1+ SWAP DO CR I 5 .R
                   I I M* I DN* 10 D.R
                LOOP ;
```

DECIMAL ---

Fixe la base à dix. Voir chapitre 5.

DPL --- adr

Variable mémorisant la position du point décimal (en fait le nombre de chiffres à la droite du point) lors de la dernière entrée numérique d'un nombre 32 bits. DPL contient −1 si le dernier nombre entré n'est pas un nombre 32 bits (si cette entrée était en réponse à INPUT ou LINPUT, DPL vaudra invariablement −1. Il n'est positionné que par l'entrée des nombres en mode interactif.).
L'utilisation de cette variable est laissée au programmeur. Il est possible, par exemple, de réaliser des formatages des affichages de manière à simuler des calculs décimaux en virgule fixe.
Exemple : calcul de TVA sur une somme pouvant être entrée sous plusieurs formes...

```
: CTS ( ud --- ud ou un --- ud. Convertit l'entrée en centimes)
  DPL @
  CASE
      -1 OF 100  U*     ENDOF        ( somme en francs, sans virgule)
       0 OF 100  UDN*   ENDOF          ( Nombre terminé par un point)
       1 OF 10   UDN*   ENDOF        ( Un seul chiffre après la virgule)
       2 OF             ENDOF           ( Entrée : Francs et centimes)
       ." Erreur" BELL ABORT              ( Dans les autres cas .....)
  ENDCASE ;
```

Puis :

```
: TVA  ( ud --- ou un ---)
     CTS 186 UDN* 1000 M/MOD
     <# # # 46 HOLD #S #> TYPE DROP ;
      1005 TVA 186.93 OK  100.5 TVA 18.69 OK
```

EMIT c ---

Le mot de base de tout l'affichage en FORTH. EMIT envoie le caractère de code ASCII c à l'écran, ou plus généralement au périphérique actuellement sélectionné en sortie.

```
65 EMIT A OK
```

127 EMIT affiche un carré en vidéo inverse (voir table des codes ASCII, en annexe).
EMIT exécute le mot (EMIT) en mode "CONSOLE", et (PEMIT) en mode "PRINTER".

En fait, EMIT exécute le mot dont l'adresse est en position 1 dans le tableau SYSVEC. Cette adresse peut être changée par les mots PRINTER (qui y place l'adresse de (PEMIT)) ou CONSOLE (pour revenir à (EMIT)), ou encore par vous, si vous désirez modifier la routine d'affichage.
Rendons notre ordinateur bègue...

```
: XX (c ---) DUP (EMIT) (EMIT) ;                ( Nouveau mot d'affichage)
: BEGUE (---) ' XX CFA 1 SYSVEC ! ;             ( ...à la place de (EMIT)...)
BEGUE OOKK
```

...Et tous les caractères à afficher apparaîtront dorénavant en double exemplaire, même l'écho lors de la frappe d'une touche. Tapez par exemple : VLIST ...
Il s'affiche VVLLIISSTT, et les noms du dictionnaire sont affichés avec des caractères doublés. A l'affichage près, tout fonctionne encore comme auparavant !!!
Pour revenir à la normale, tapez : ' (EMIT) 1 SYSVEC !
qui donnera à l'écran : "((EEMMIITT) CCFFAA 1 1 SSYYSSVVEECC !! OK
Pour vos essais, vous aurez intérêt à définir un mot qui revienne à l'affichage normal avant d'exécuter le mot BEGUE... Il vaudrait mieux que le nom de ce mot ne soit pas trop compliqué à taper... Ce pourrait être par exemple un nom en une seule touche...
Pourquoi pas la touche < STOP > ?... Il suffit de taper les touches ":", espace, < STOP >, espace, puis l'action à exécuter (voir ci-desssus).... Pour l'exécuter, tapez la touche < STOP > entre des espaces (ou < ENTREE >).

EXPECT adr n ---

Mot système. EXPECT est le mot de bas niveau qui réalise la lecture de chaînes de caractères au clavier en FORTH. Il lit n caractères (ou jusqu'à la frappe de la touche ENTREE), et les dépose à partir de l'adresse adr, suivis par un ou deux caractères de code ASCII Ø.
LINPUT$, INPUT$, etc. utilisent EXPECT (adr étant donné par PAD), ainsi que QUERY (adr étant donné par TIB @).

HEX ---

Fixe la base à seize (système hexadécimal). Voir chapitre 5.

IN? --- adr

Mot système. IN? donne l'adresse du prochain caractère dans le flot d'entrée (prochain caractère à traiter par l'interpréteur, le plus souvent). C'est à partir de cette adresse que WORD (voir chapitre 5) commence la recherche du prochain mot.

INKEY --- c

Lecture non bloquante du clavier. INKEY teste si une touche est actuellement enfoncée, et laisse sur la pile la valeur suivante :
— Le code ASCII de la touche (si une touche est enfoncée).
— La valeur Ø dans le cas contraire.
Le code ASCII Ø étant inaccessible au clavier, ce mot peut souvent remplacer ?TERMINAL. Il est souvent utile pour les applications graphiques nécessitant la lecture "au vol" d'un caractère (ex : mobile qui se déplace en fonction des touches appuyées, et qui continue son mouvement si on ne touche pas au clavier).
Pour tester INKEY :

```
: TEXT-INKEY BEGIN INKEY ?DUP
                  IF EMIT ELSE BELL THEN
                  PTRIG UNTIL ;
```

Noter l'utilisation de ?DUP, qui dépile la valeur Ø dans les cas où aucune touche n'est enfoncée.

INPEN --- x y

Lecture non bloquante des coordonnées du point visé par le crayon optique. INPEN laisse sur la pile les coordonnées graphiques du point visé (x entre Ø et 319, y entre Ø et 199. Voir chapitre relatif au graphique), et (– 1 – 1) en cas de lecture erronée. Suivant le réglage de la luminosité, vous pouvez placer le crayon optique jusqu'à sept ou huit centimètres de l'écran.
Rappelons que pour des raisons techniques, la lecture sera erronée si le point visé est de couleur rouge ou noire. D'autres cas de lecture invalide peuvent être une visée hors-écran, ou le crayon optique trop éloigné.

Exemple : dessiner au crayon optique...

```
: TRACER ( ---) INPEN DUP Ø <
                IF DDROP ELSE LINETO THEN ;
: DESSIN ( ---) INPUTPEN PSET
               BEGIN TRACER PTRIG UNTIL ;
```

Puis : DESSIN (Promenez votre crayon optique devant l'écran)
(Appuyez une fois pour démarrer, une fois pour arrêter...)

Voir aussi l'exemple du mot LINETO, au chapitre 6.

INPUT --- d

Attend l'entrée d'une chaîne de caractères au clavier (terminée par la touche ENTREE), qui est convertie en un nombre 32 bits, dont la valeur vient sur la pile. La conversion s'arrête sur le premier caractère non convertible, compte tenu de la base utilisée.

```
INPUT D.    ( Tapez 1234,56)  1234  OK
INPUT D.    ( Tapez -12345)  - 12345 OK
INPUT D.    ( Tapez AZERTY)  Ø OK
HEX INPUT D. DECIMAL  ( Tapez 1AR3)  1A  OK
                              ( Le caractère R n'est pas un symbole hexadécimal)
2 BASE ! INPUT D.  ( Tapez 121Ø)  1  OK
                                   ( 2 n'est pas un chiffre en base 2)
```

INPUT$ --- str

Lit une chaîne de caractères à partir du périphérique actuellement positionné en entrée (en général le clavier). La chaîne obtenue comporte au maximum 8Ø caractères, et si on tape effectivement 8Ø caractères, la saisie prendra fin au quatre-vingtième caractère, sans qu'il soit besoin de taper la touche ENTREE.
En fait, INPUT$ est défini de la manière suivante :

```
: INPUT$   8Ø  LINPUT$  ;  (voir ce mot).
```

En particulier, prenez garde à sauvegarder au plus tôt la chaîne saisie, si vous voulez la réutiliser plus tard.
Voir LINPUT$

INPUTPEN --- n1 n2

Réalise la lecture bloquante des coordonnées du point de l'écran que vise le crayon optique. Il est nécessaire d'appuyer le bouton du crayon optique pour valider la lecture (de ce fait, INPUTPEN peut être utilisé, comme KEY, pour arrêter une application). Voir aussi INPEN ou PTRIG.
Aucune vérification n'est faite quant à la validité de la lecture, au programmeur de s'en occuper si nécessaire...

```
: LIRE 0. BEGIN DDROP INPUTPEN DUP 0< NOT UNTIL ;
```

Noter le 0. (dépilé immédiatement par DDROP) pour initialiser la pile. Vous pouvez d'ailleurs le remplacer par n'importe quel nombre double longueur... ou tout objet occupant deux octets sur la pile.

KEY --- n

Réalise la lecture (bloquante) d'un caractère sur le périphérique d'entrée (en général le clavier). KEY laisse sur la pile le code ASCII du caractère tapé, et n'affiche pas le caractère (si vous voulez que la touche tapée apparaisse à l'écran, il faut le prévoir. Cela se fait en général par "DUP EMIT").
Les séquences de caractères accentués sont respectées, et traduites par un code unique (voir en annexe la liste des caractères accessibles au clavier). Cela signifie que si vous commencez par appuyer sur la touche ACC, FORTH attendra deux autres frappes (sauf bien entendu si la deuxième touche tapée ne fait pas partie d'une séquence de caractère accentué, auquel cas cette touche est acceptée, et la touche ACC n'est pas prise en compte).
KEY est souvent utilisé pour stopper momentanément l'exécution d'une application, en attendant la frappe d'une touche...

```
DESSIN KEY DROP CLS
```

... Vous permet de contempler une dernière fois votre oeuvre avant que le CLS ne la fasse disparaître.

```
KEY DUP . EMIT
```

Vous évitera d'aller consulter une table pour trouver le code ASCII d'un caractère.

LINPUT n --- d

Attend l'entrée d'une chaîne de caractères de n chiffres au maximum en provenance du périphérique sélectionné en entrée (en général le clavier). Cette entrée sera interprétée comme un nombre 32 bits d, dont la valeur sera déposée sur la pile.
La saisie s'arrête immédiatement après la frappe du n-ième chiffre (sans validation par la touche ENTREE), ou avant le n-ième chiffre par appui sur la touche ENTREE.

```
1 LINPUT DROP
```

Attend la frappe d'un seul chiffre (en l'affichant à l'écran) et laisse sa valeur sur la pile (comme nombre simple longueur, grâce à DROP). LINPUT utilise la base actuellement sélectionnée pour la conversion en nombre.

```
2 BASE ! 1 LINPUT CR D. (Tapez 3) 0 OK
```

Le caractère "3" n'est pas un chiffre en base 2. Le résultat de la conversion est donc Ø...

LINPUT$ n --- str

Réalise la lecture d'une chaîne de n caractères au maximum (moins si on a tapé la touche ENTREE avant le n-ième caractère) en provenance du périphérique actuellement positionné en entrée (le plus souvent le clavier), avec écho sur le périphérique positionné en sortie (en général l'écran).
Si on tape effectivement n caractères, la saisie s'arrête au n-ième caractère sans qu'il soit besoin de taper la touche ENTREE.
La chaîne lue est mise dans une zone temporaire (à l'adresse donnée par PAD, pour être précis), et doit être utilisée avant que cette zone serve de nouveau (avant un autre INPUT$, INPUT, LINPUT, ou une définition de chaîne temporaire par le mot " (guillemet), entre autres...). Le moyen le plus sûr est de ranger immédiatement la chaîne dans une variable.

```
20 STRING NOM 20 LINPUT$ NOM $!
                        ( Sauvegarde immédiate de la chaîne dans la variable NOM)
PRINTER 10 LINPUT$ CR CONSOLE 2 LEFT$ TYPE
```

Lit 1Ø caractères au clavier, les affiche à l'imprimante, puis affiche à l'écran les deux premiers caractères de la chaîne tapée. Le CR force l'imprimante à vider son buffer (pour afficher immédiatement).

ONPEN --- n

Laisse sur la pile le numéro de la zone (précédemment définie par BOXPEN ou PEN) pointée par le crayon optique, ou le nombre – 1 si le crayon optique pointe sur une zone non précédemment définie. Cette lecture est bloquante, mais ne teste pas la validité des résultats. De sorte qu'il est commode de faire une boucle pour attendre une lecture valide, ou de considérer toutes les zones non définies comme une "zone – 1".
ONPEN est idéal pour la gestion des choix par le crayon optique (menus), souvent utilisé en liaison avec CASE ou GODO.
Exemple : si on a défini les zones Ø, 1, 2, 3, 4, et les options Ø, 1, 2, 3 (mots OPTØ, OPT1,...).

```
: CHOIX   BEGIN ONPEN >R R@ -1 > R@4 < AND\ND
            UNTIL ( Lit jusqu'à ce qu'il trouve une zone valide)
            R>  GODO OPTØ OPT1 OPT2 OPT3 THEN ;
```

Ou

```
: CHOIX ONPEN 1+  GODO BELL OPTØ OPT1  OPT2  OPT3 BELL THEN ;
                        ( GODO prend des paramètres à partir de Ø, et non de -1)
```

Puis bien sûr :

```
: MENU BEGIN DEFINIT-ZONES AFFICHE-TEXTE CHOIX AGAIN ;
```

PRINTER ---

Sélectionne le port de communication comme port de sortie (parallèle). Après cette sélection, EMIT enverra les caractères à afficher vers l'imprimante (si c'est une imprimante qui est connectée), ou vers tout périphérique connecté au port de communication. Voir aussi le mot CONSOLE, qui rétablit l'écran comme périphérique de sortie. Le principe de cette vectorisation est expliqué en annexe.
En mode "PRINTER", les retours-chariot (code ASCII 13) sont transformés en "Retour-chariot + Passage à la ligne" (codes ASCII 13 et 1Ø).

```
8Ø C/L ! PRINTER VLIST CONSOLE
```

PTRIG --- cond

Laisse sur la pile la valeur 1 si le bouton du crayon optique est enfoncé, Ø dans le cas contraire.
Souvent utilisé comme condition d'arrêt dans les applications graphiques, ou lorsqu'on veut garder la disponibilité du clavier pour des entrées. Voir aussi ?TERMINAL et BUTTON

```
: XXX ... BEGIN ... PTRIG UNTIL ... ;
```

(Répéter jusqu'à ce qu'on appuie sur l'extrêmité du crayon optique...).

QUERY ---

Mot système qui réalise la lecture de 8Ø caractères au clavier (ou moins, si on a tapé la touche ENTREE), et les range dans le buffer d'entrée du clavier (voir TIB). Le mot WORD (voir chapitre 5) peut être utilisé pour obtenir du texte (mot par mot) de ce buffer, en positionnant les variables > IN et BLK à Ø.

SCRPRT ---

Réalise une "hard-copy" graphique de l'écran sur l'imprimante. D'origine, ce mot est prévu pour fonctionner avec l'imprimante thermique THOMSON PR90-040.
Mais il est vectorisé, de sorte que pour une autre imprimante, l'utilisateur peut écrire un mot qui réalise cette copie graphique, puis mettre son cfa en position 11 dans le tableau SYSVEC (voir annexe D pour plus de détails sur cette possibilité).

SPACE ---

Affiche un espace sur le périphérique de sortie (écran, imprimante ou autre).
SPACE est en fait défini par 32 EMIT.

SPACES n ---

Affiche n espaces sur le périphérique de sortie (écran, imprimante ou autre).

```
CR INVCOLOR 40 SPACES INVCOLOR
```

STICK n1 --- n2

Lit la position des manettes de jeux. n1 est le numéro de la manette à lire (Ø ou 1), et le résultat n2 correspond à la position du "manche à balai". n2 est un nombre de Ø à 8 (Ø si la manette est verticale).
Exemple : déplacement d'un dessin par les manettes de jeux.

```
: DEPLACE ( ---)
        BEGIN Ø STICK
              GODO SUR-PLACE NORD NORD-EST EST SUD-EST SUD
                   SUD-OUEST OUEST NORD-OUEST THEN
        PTRIG UNTIL ;
```

A vous, bien sûr, d'écrire les mots NORD, EST, ... (Pour indication, le mot SUR-PLACE pourrait bien être : SUR-PLACE ;)
NORD devrait consister en : effacer le dessin, et le redessiner un (ou plusieurs) point plus haut. De même pour les autres déplacements.

TIB --- adr

Mot système. TIB est une variable contenant l'adresse du buffer d'entrée du terminal (pour Terminal Input Buffer). Cette zone contient ce que vous tapez au clavier lorsque vous entrez du texte FORTH. Elle comporte 8Ø caractères (longueur maximum d'une entrée du clavier).
Le texte entré se termine par un ou plusieurs codes Ø. Chaque frappe de la touche ENTREE a pour effet de faire traiter le buffer d'entrée par l'interpréteur, puis de le "vider".

```
: VOIRTIB CR TIB @ BEGIN DUP C @ DUP
                  WHILE EMIT ." -" 1+ REPEAT DROP ;
```

Puis : BELL BELL RED FRAME VOIRTIB
réaffiche votre entrée après l'avoir traitée...

TYPE **str ---**

Le mot standard d'affichage d'une chaîne de caractères. TYPE est construit à partir de EMIT (voir ce mot), donc peut envoyer une suite de caractères à n'importe quel périphérique en sortie (le plus souvent écran ou imprimante, mais en fait tout périphérique connecté au port de communication si on a utilisé PRINTER, ou même cassette ou disque si on a remplacé (EMIT) dans le tableau SYSVEC).
A l'écran, les chaînes affichées par TYPE peuvent contenir des caractères graphiques, mais l'imprimante imprimera ce qu'elle pourra... (Voyez dans la documentation de votre imprimante comment sont affichés les caractères dont le code ASCII est supérieur à 128...).

```
PRINTER PHRASE TYPE CR CONSOLE
```

U. **un ---**

Affichage non signé d'un nombre 16 bits. Le nombre sur la pile est seulement interprété comme non signé pour l'affichage (pour la représentation sur la pile, FORTH ne fait aucune différence entre signé et non signé. C'est à l'utilisateur de décider de l'usage qu'il fait des nombres de la pile).
Visualisation de la représentation des nombres de la pile...

```
: UB. 2 BASE ! U. DECIMAL ;
126 UB. 1111110 OK
32767 UB. 111111111111111 OK
-32767 UB. 1000000000000001 OK ( Complément à deux)
```

U.R **n p ---**

Affiche le nombre non signé n justifié à droite dans une zone de p espaces. U.R est à U. ce que .R est au mot "." (point).
Voir .R ... En remplaçant dans CARRES le deuxième mot .R par le mot U.R, le domaine de validité du mot devient : $-255 < n1 < n2 < 255$ (les nombres non signés vont de 0 à 65535, alors que les nombres signés ne dépassent pas 32767).

UD. d ---

Affiche un nombre 32 bits non signé (suivi d'un espace).
UD. remplit simultanément les fonctions de U. et de D. (non signé comme U., et 32 bits comme D.).

```
2 BASE ! -1. UD. 11111111111111111111111111111111 OK
```

UD.R d p ---

Affichage d'un nombre 32 bits non signé d, justifié à droite dans une zone de p espaces. UD.R remplit simultanément les fonctions de UD. et .R (32 bits non signé comme UD., et justifié à droite comme .R). Voir D.R (remplacer dans le mot CUBES le mot D.R par le mot UD.R. Le domaine de validité est modifié).

Chapitre 10

Stockage de masse : cassettes et disquettes

Les données sont rangées sur le stockage de masse par unités de 1 k-octet, qu'on appelle des "blocks". FORTH communique avec les cassettes ou les disquettes par l'intermédiaire de buffers, qui sont situés en haut de la mémoire (voir organisation mémoire en annexe E). Il est possible de choisir le nombre de buffers en mémoire (voir MAX-BUF, #BUF, BUFFERS). En version disque, cette communication doit en principe rester transparente pour l'utilisateur. Le mot fondamental est le mot BLOCK, qui lit un block de la disquette, et le range dans un buffer, à la place d'un autre. Mais si le block qui occupait ce buffer avait été modifié depuis sa lecture, il sera réécrit sur disque auparavant (il appartient en général au programmeur de décider si un buffer doit être marqué comme modifié, par le mot UPDATE). En version cassette, cette méthode est évidemment impraticable, du fait de la nécessité de positionner manuellement la bande. Un block modifié ne sera pas réécrit sur cassette de manière systématique, le programmeur est responsable d'une éventuelle sauvegarde (par SAVE-BUFFERS).

Les blocks peuvent servir à ranger, entre autres données, les textes écrits avec l'éditeur (le plus souvent les sources de vos applications, mais aussi peut-être des fichiers ASCII...). On parle alors plutôt d'"écrans de l'éditeur". La notion importante à ce niveau est la notion de flot d'entrée. L'interpréteur FORTH n'est pas limité à traiter des

informations (un programme...) à une adresse fixe, il peut traiter du texte provenant soit d'un block, soit du buffer des entrées au clavier (voir TIB, au chapitre précédent).
La variable BLK le renseigne sur le numéro de block, ou encore d'écran, à interpréter (si BLK contient Ø, le flot d'entrée est le clavier. Il est donc impossible qu'un block d'une disquette ou d'une cassette ait le numéro Ø).
Enfin, la mémoire de masse peut contenir des sauvegardes-dictionnaire. Seule la partie RAM-utilisateur est alors sauvegardée. Il est donc possible de sauvegarder une application compilée, de la rappeler pour l'exécuter (voir SAVE et RUN). Il est aussi possible de fixer un mot particulier comme devant être exécuté automatiquement au chargement (voir AUTORUN). Toutefois, ces mots étant plutôt à considérer comme des commodités en regard du Standard FORTH, vous les trouverez dans le chapitre 11 (mots divers).
La cartouche-ROM contient des mots tenant compte de la possibilité d'utilisation de disquettes, mais ces mots ne seront opérationnels en version disque que si le système d'exploitation est chargé en mémoire. Dans ce cas, le copyright qui apparait lors de l'initialisation est :

FORTH-TO7 DISK 1.Ø. (ou MO5 au lieu de TO7).

Le chargement du système d'exploitation se fait lors de la mise sous tension (voir annexe F pour les modalités de démarrage). Il s'accompagne du chargement d'un complément de dictionnaire, composé de mots permettant la gestion du disque.
Ces mots sont décrits dans l'annexe B. Ce chapitre ne décrit que les mots présents dans la cartouche-ROM, même si certains d'entre eux n'ont pas la même action en version disque qu'en version cassette, et si certains exemples concernent l'emploi de disquettes.
Si vous utilisez des disquettes, il convient donc de compléter les mots de ce chapitre par ceux de l'annexe B...

Ce chapitre comporte aussi des mots plus ou moins techniques, permettant de commander de manière "personnalisée" le port de communication, le lecteur-enregistreur de cassettes, et le contrôleur des lecteurs de disquettes (en particulier pour réaliser la lecture ou l'écriture caractère par caractère, ou pour communiquer avec un périphérique quelconque...).
Ces mots font partie des possibilités du matériel, mais la plupart des utilisateurs n'auront probablement jamais à s'en servir...
Si vous voulez les employer, il est indispensable de respecter soigneusement les protocoles indiqués.

l'écriture caractère par caractère permet la création de fichiers de tailles variables (pas seulement par blocks de 1 k). Il est alors fondamental de ne pas oublier d'activer le contrôleur du périphérique concerné avant opération, ni de le désactiver après le travail (voir IO, (DIO), (KIO), (PIO)). Pour les disquettes, cette opération cherche ou crée une entrée dans le catalogue de la disquette, puis (après travail) vide le buffer d'entrées-sorties (voir DMA) si on est en écriture. Pour les cassettes, cette opération cherche ou écrit un bloc de début de données (suite de codes ayant un format spécial, permettant au système de reconnaître le début d'un fichier), puis, en fin de travail, vide le buffer des entrées-sorties avant d'écrire sur la cassette un bloc de fin de données (en écriture). Pour le port de communication, c'est l'opération d'activation qui décide du mode de fonctionnement (série ou parallèle, lecture ou écriture...).

En outre, la plupart des mots qui gèrent les entrées-sorties sur le stockage de masse sont vectorisés par l'intermédiaire du tableau SYSVEC, pour permettre l'adaptation de nouveaux périphériques. Dans ce chapitre, cette possibilité est seulement mentionnée pour les mots concernés.
Pour plus de détails sur son utilisation, reportez vous à l'annexe D.

#BUF --- n

Laisse sur la pile le nombre actuel de buffers utilisés par le système. A la mise sous tension le langage teste la RAM. Si vous disposez de 8 k ou moins, il allouera 2 buffers. Dans le cas contraire, vous aurez 4 buffers.

```
#BUF . 4 OK
```

Voir aussi les mots MAXBUF et BUFFERS.

(FR/W) adr n cond ---

Mot système réalisant le transfert de données par blocks entre disquette ou cassette, et mémoire.
cond indique le sens du transfert (Ø pour une écriture, 1 pour une lecture).

n est le numéro du block de la disquette ou de la cassette (à lire ou à écrire).
adr est l'adresse mémoire où se situent les données à écrire (ou les données lues) sur le stockage de masse.
Le block est sauvegardé (ou lu) avec le nom "BLOCKn.DAT".
(FR/W) utilise les mots IO, PUTC et GETC, et peut donc, moyennant modification de SYSVEC (voir vectorisation de IO, GETC, PUTC), lire ou écrire sur d'autres périphériques.
(FR/W) est lui-même un mot appelé par défaut par le mot R/W (via SYSVEC), ce qui permet aussi de personnaliser la gestion des périphériques.

(KBUF) n --- adr

Mot système appelé par BUFFER en version cassette.
(KBUF) réalise le même travail que (BUFFER), à cela après qu'il ne sauvegarde pas un block modifié dans le cas où le buffer correspondant doit être réalloué.

(KEMIT) c ---

Mot système réalisant l'écriture d'un caractère sur cassette.
Avant d'écrire le premier caractère d'un ensemble de données sur cassette, il faut initialiser l'opération d'écriture par 2 (KIO).
Après avoir fini d'écrire, il faudra taper Ø (KIO) pour terminer l'opération, et vider le buffer d'entrées-sorties de la cassette.
La variable D-ERR permet de savoir si l'opération s'est bien déroulée.
(KEMIT) est appelé par le mot PUTC (voir ce mot) via le tableau SYSVEC (position 17).

(KFLUSH) ---

Mot système réalisant l'écriture sur cassette (ou disquette) du dernier block référencé (voir la variable PREV). A la différence de (FLUSH), seul le dernier block est recopié (et le bit d'update est remis à Ø).

(KIO) n ---

Initialise le lecteur-enregistreur de cassettes pour une opération d'entrée-sortie. Le nombre n indique le type d'opération à réaliser. Avant d'utiliser (KIO), il faut avoir correctement affecté les variables DNAM (adresse du nom de fichier), et DMA (adresse du buffer tampon utilisé pour les entrées-sorties cassette. En écriture, ce buffer doit contenir 128 octets au moins, et en lecture 255 si on désire lire des données créées sous BASIC). Les types d'opération sont les suivants :

n = Ø Termine une opération d'entrée-sortie. En lecture, désalloue les pointeurs vers les données. Dans le cas d'une écriture, Ø (KIO) écrit sur cassette les données restantes du buffer d'entrées-sorties, puis écrit un bloc de fin de données.

n = 1 Initialise une opération de lecture. Recherche sur la cassette un bloc d'en-tête dont l'identificateur correspond aux 11 premiers caractères du nom positionné (pointé par la variable DNAM).

n = 2 Initialise une opération d'écriture. Ecrit sur la cassette un bloc d'en-tête en utilisant le nom pointé par la variable DNAM... Après exécution de ce mot, la variable D-ERR permet de savoir si l'opération s'est bien passée. (KIO) est appelé par le mot IO, via le tableau SYSVEC (position 15).

(KKEY) --- c

Mot système utilisé pour lire caractère par caractère sur la cassette. le code ASCII du caractère lu est déposé sur la pile. En fait, un nombre sera toujours déposé sur la pile, même en cas d'erreur de lecture. Il appartient au programmeur de tester la validité du caractère lu en testant la variable D-ERR.
Avant d'utiliser (KKEY) pour lire une suite de données, il faudra initialiser l'opération de lecture par 1 (KIO).
Si on désire arrêter la lecture d'un ensemble de données avant la fin (avant la rencontre d'un bloc de fin de données), il suffit d'exécuter Ø (KIO).

(PIO) **n ---**

Mot système précisant le mode de fonctionnement du port de communication. n indique le type d'opération à réaliser.

n = Ø Termine l'opération d'entrée-sortie en cours.

n = 1 Initialise le port de communication en mode série (RS – 232) lecture/écriture.

n = 2 Initialise le port de communication en mode série (RS – 232) écriture seule.

n = 3 Initialise le contrôleur de communication en mode parallèle (centronics). C'est un mode "écriture seule" (le plus souvent, l'imprimante).

n = 4 Initialise la copie d'écran graphique... Après utilisation de (PIO), la variable D-ERR permet de savoir si l'opération s'est bien passée.

+BUF **adr1 --- adr2 cond**

Mot système de base pour parcourir les buffers en mémoire. A partir de l'adresse adr1 d'un buffer donné, +BUF calcule l'adresse adr2 du buffer qui suit (physiquement), en mémoire. Si adr1 est l'adresse du dernier buffer (celui d'adresse la plus élevée), +BUF donne l'adresse du premier buffer (connue par FIRST).
cond est une valeur de vérité indiquant si oui ou non le buffer d'arrivée (adresse adr2) est le dernier référencé (voir PREV).
+BUF est utilisé par tous les mots du système qui font intervenir plusieurs buffers (ex : (FLUSH), (BUFFER), (FR/W),...) et bien entendu par les mots qui les appellent...

B/BUF **--- n**

Une constante donnant le nombre 1Ø24. Il s'agit en fait du nombre d'octets par block (en anglais bytes/buffer).

```
15 BLOCK B/BUF DUMP (Affiche le contenu du block 15 (dump hexadécimal)).
```

BBLOCK n --- adr

Laisse sur la pile l'adresse du premier octet utilisable (début effectif du block, voir BLOCK) du buffer correspondant au block n. Si le block n'est pas actuellement dans un buffer, BBLOCK lui alloue un buffer qu'il met à blanc (remplit avec des caractères "espace").
La raison d'être de ce mot est la gestion des blocks de la cassette. Il n'est pas acceptable en effet d'utiliser le mot BLOCK dans ce cas, puisque toute la cassette se déroulerait lorsqu'on voudrait créer un nouveau block.
Si on veut lire un block existant sur cassette, on peut bien sûr se servir de BLOCK. Encore faudra-t-il positionner correctement la bande (avant le fichier correspondant).
L'éditeur utilise BBLOCK en version cassette, ce qui oblige, pour lire un block, à taper explicitement CNT-R (pour forcer la lecture de la cassette) après avoir créé le block (ex : 15 EDIT puis, sous l'éditeur : CNT-R).

BLK --- adr

Variable système contenant le numéro du block positionné comme flot d'entrée pour l'interpréteur. Si le contenu de BLK est Ø, le flot d'entrée considéré proviendra du clavier.
Cela implique en particulier qu'un block est inutilisable pour l'interpréteur. Par exemple, "Ø EDIT" vous éditera bien un block Ø, mais vous ne pourrez pas l'exécuter.

BLOCK n --- adr

Un des mots les plus couramment utilisés pour utiliser la mémoire de masse pour traiter des données. BLOCK dépose sur la pile l'adresse du début de la zone de stockage du buffer contenant en RAM le block numéro n.
En fait, les buffers comportent 2 octets d'en-tête (contenant le numéro de block, et le bit d'update), suivis des 1Ø24 octets (c'est l'adresse du premier octet de cette zone que donne le mot BLOCK), et sont terminés par 2 octets à Ø (qui servent à marquer la fin du flot d'entrée lorsqu'on interprète le block).
Si le block n n'est pas actuellement en mémoire, BLOCK lui fait allouer un buffer (par le mot BUFFER). En version disque, si le buffer à utiliser (voir USE) est marqué comme modifié, il est réécrit sur disquette avant le chargement du block n. En version cassette, l'opération est manuelle, et l'utilisateur aura à forcer la réécriture du block (par exemple par SAVE-BUFFERS, ou par (KFLUSH)), s'il le désire, avant d'utiliser le mot BLOCK.
Exemple d'utilisation : fichier ASCII, fiches de longueur fixe (L), mémorisé dans des blocks consécutifs à partir d'un block de numéro DEBUT (constante).

```
43 CONSTANT DEBUT 13 CONSTANT L L STRING NOM
: FICHE                    ( n --- Charge la fiche et laisse son adresse de début)
      1- L *                    ( Nombre d'octets depuis le début de fichier )
      B/BUF /MOD DEBUT + BLOCK          ( Charge le block concerné...)
      + ;                              ( Donne l'adresse de début de la fiche )
```

Puis par exemple :

```
5 FICHE L TYPE                                    ( pour afficher la fiche...)
NOM 5 FICHE SWAP CMOVE                            ( pour modifier la fiche...)
```

(Attention, ici le mot $! utiliserait une longueur maximum de mot, qui n'existe pas dans cette manière de procéder. Une autre façon serait de structurer le fichier comme un tableau de chaines de caractères, chaque enregistrement ayant un octet de longueur maximum et un octet de longueur).

BUFFER n --- adr

C'est ce mot qui réalise la première partie du travail du mot BLOCK. Il alloue un buffer au block n, et sauvegarde éventuellement l'ancien contenu de ce buffer s'il était marqué comme modifié (mais il ne lit pas le block n).
BUFFER exécute en fait le mot dont l'adresse est en position 13 du tableau SYSVEC. En général, on y trouve l'adresse de (KBUF) si on n'utilise pas de disquette (version cassette, ou disque non branché) , ou l'adresse de (BUFFER) si on se sert de disquettes. Il est possible, si vous voulez modifier cette gestion de créer un autre mot et de mettre son adresse dans SYSVEC.
Exemple : un mot qui demande confirmation pour sauvegarder...

```
: XBUF Ø 24 LOCATE ." Voulez vous sauvegarder le block "
       PREV @ @ 127 AND . ." ?"
                                   ( 127 AND tient compte du bit d'update)
       BELL BELL KEY " O" ASC = (A-t-on tapé un "O" ?)
       IF (BUFFER) ELSE (KBUF) THEN ;
FIND XBUF 13 SYSVEC !                    ( Remplace BUFFER par XBUF)
```

Pour tester l'effet, choisissez par exemple 2 buffers, et tapez :

```
12 BLOCK 13 BLOCK 14 BLOCK 12 BLOCK 13 BLOCK 14 BLOCK 12 BLOCK ...
```

Chaque fois que votre appareil aura besoin de sauvegarder un block, il vous demandera confirmation. Si vous tapez la touche "O", votre lecteur de disquettes (ou votre magnétophone à cassettes) tournera nettement plus longtemps (sauvegarde + lecture au lieu de sauvegarde seule).
Pour revenir à la situation par défaut :

```
FIND (BUFFER) 13 SYSVEC !          ( si vous êtes en version disquette)
FIND (KBUF) 13 SYSVEC !            ( si vous êtes en version cassette)
```

BUFFERS n ---

Permet de choisir le nombre de buffers de blocks disponibles. Quelle que soit la valeur choisie pour n, le nombre de buffers ne peut être inférieur à 2, ni supérieur à MAXBUF.

```
2 BUFFERS ( Allouera 2 buffers)
```

Attention, BUFFERS réinitialise les buffers. Leur contenu antérieur est perdu. Il sera donc prudent, auparavant, de les sauvegarder en utilisant SAVE-BUFFERS ou, sous l'éditeur, la commande CNT-W...

D-ERR --- adr

Variable d'un octet contenant le code de "compte-rendu" après une opération d'entrée-sortie. Attention, D-ERR est une variable en un seul octet (variable 8-bits, ou C-variable)... (Autrement dit, on y accède par C@, et non pas par @).
Les valeurs possibles de D-ERR sont les suivantes :

Ø — L'opération s'est bien déroulée.
1 — Code erroné dans une instructio:
2 — Erreur de lecture.
3 — Erreur d'écriture.
4 — Fichier non trouvé dans le catalogue de la disquette.
5 — Disque plein (ou catalogue plein).
8 — Le fichier existe déjà (tentative de réécriture).
255 — Fin de données (lecture cassettes ou disquettes. Dans ce cas, le dernier caractère lu est erroné).

La plupart des mots qui réalisent des entrées-sorties positionnent D-ERR, mais n'ont pas un comportement spécial en cas d'erreur. A l'utilisateur de tester D-ERR pour réaliser son propre traitement d'erreur. La programmation en est certes alourdie, mais les avantages sont importants (traitement personnalisé des erreurs).

DMA --- adr

Variable contenant l'adresse du buffer des entrées-sorties pour la lecture ou l'écriture sur cassettes ou disquettes.
Toutes les informations qui transitent entre l'ordinateur et le stockage de masse passent par ce buffer de 256 octets (ou 128 pour les disquettes en simple densité). Voir en annexe E le schéma d'implantation mémoire du FORTH.

DNAM --- adr

Variable contenant l'adresse du nom du fichier qui sera utilisé pour les prochaines opérations de lecture-écriture. Ce nom occupe 11 octets, 8 pour le nom proprement dit et 3 pour l'extension. Vous pouvez positionner vous-même ce nom si vous voulez lire ou écrire un fichier BASIC ou autre, mais en principe FORTH gère les noms des fichiers qu'il emploie sans que vous ayez à vous préoccuper de leurs noms.

Les écrans de l'éditeur sont des fichiers de noms BLOCKxxx.DAT
Les sauvegardes-dictionnaires sont des fichiers de noms DICTxxx.FTH
Dans les deux cas, xxx est un nombre (de 3 chiffres maximum pour un écran de l'éditeur), et le point qui sépare le nom de l'extension n'est ici que pour la lisibilité. Il n'est pas à taper au clavier... Voir annexe E. Pour fixer le nom du fichier à utiliser, employez FNAM! et FNB!.

DOS ---

Pour MO5 uniquement. Ce mot demande à FORTH de se réinitialiser en tenant compte de la présence d'un lecteur de disquettes.
DOS s'emploie après la mise sous tension, pour charger le système d'exploitation (voir annexe B) ou en cours de travail pour redémarrer à zéro.

EMPTY-BUFFERS ---

Vide les buffers actuellement en mémoire. Le contenu antérieur des buffers est perdu.
Si vous avez modifié un block (voir UPDATE) en mémoire, et si vous utilisez des disquettes, il est vigoureusement conseillé de taper EMPTY-BUFFERS lors de tout changement de disquette.
Si par exemple vous avez, sous l'éditeur, écrit dans l'écran 1Ø (et donc modifié le block 1Ø), lorsque le système aura besoin d'allouer le buffer qui contient ce block, il le réécrira sur la disquette... Si entretemps vous avez changé de disquette, le block 1Ø s'écrira sur votre nouvelle disquette, effaçant éventuellement un block 1Ø qui s'y trouverait... La solution est de vider les buffers pour protéger le contenu de vos disquettes.
Evidemment, si vous voulez copier le block 1Ø sur une autre disquette, ne faîtes pas d'EMPTY-BUFFERS !!!

EOF --- cond

Indicateur laissant sur la pile la valeur "vrai" (1) si le fichier en cours de lecture est terminé. Ce mot est surtout utilisé pour les applications lisant caractère par caractère sur cassette ou disquette.
Exemple : voir IO

FASC! n ---

Permet de fixer le type de données que contiendra le prochain fichier à lire ou à écrire. La valeur de n doit être Ø ou 255. (Ø pour des données binaires, 255 pour des données ASCII).
Le type de données apparaît lors d'un DIR, dans la quatrième zone d'affichage d'un nom du catalogue (A pour ASCII, B pour Binaire). Les blocks de l'éditeur sont composés de données ASCII, alors qu'une sauvegarde-dictionnaire est composée de données binaires (code FORTH). Mais tant que vous faites du FORTH, il est inutile de vous en préoccuper.
FASC! sert lors de la lecture ou de l'écriture caractère par caractère, dans un fichier qui n'est pas un fichier standard FORTH.
Exemple : voir IO

FIRST --- adr

Laisse sur la pile l'adresse du premier buffer, celui dont l'adresse est la plus basse. En fait, FIRST laisse l'adresse du numéro du premier buffer, deux octets avant le début du buffer proprement dit.
Voir en annexe E l'organisation de la mémoire.
Pour connaître la place disponible pour l'extension du dictionnaire :

```
FIRST PAD - 255 - . 9823 OK
```

(système 24 k avec 1Ø buffers... et quelques mots déjà définis en RAM...)

FNAM! str---

Permet de fixer le nom du prochain fichier à utiliser pour une opération de lecture ou d'écriture sur cassette ou disquette. Le nom est le contenu de la chaine de caractères (str). Seuls les 11 premiers caractères seront pris en compte (8 pour le nom proprement dit, 3 pour l'extension).
FNAM! s'utilise pour lire ou écrire un fichier autre que les fichiers FORTH (qui ont pour noms BLOCKxxx.DAT ou DICTxxx.FTH), et est aussi employé par le système pour positionner les noms des fichiers FORTH. Voir aussi DNAM et FNB!.
Exemple : voir IO

FNB! n1 n2 ---

Modifie un nom de fichier à utiliser, en y plaçant le nombre n1 en position n2.
Ce mot est employé par le système pour construire les noms de fichier à utiliser (noms du type BLOCKxxx.DAT ou DICTxxx.FTH... C'est FNB! qui positionne le numéro xxx).

```
" FICHIER XYZ" FNAM!
```

Le nom est : FICHIER XYZ (1 espace entre R et X)

```
5 7 FNB!
```

Le nom est : FICHIER5XYZ (Extension XYZ)
Vérification : DNAM @ 11 TYPE FICHIER5XYZ OK

FTYP! n ---

Permet de fixer le type de fichier à utiliser (celui dont le nom est dans la zone pointée par la variable DNAM) pour la prochaine opération de lecture ou d'écriture sur disquette. Les valeurs possibles de n sont :

n = Ø	Programme BASIC	(B)
n = 1	Fichier de données (BASIC ou FORTH)	(D)
n = 2	Programme en langage machine (par exemple : sauvegarde dictionnaire)	(M)
n = 3	Fichier source en assembleur	(A)

La lettre entre parenthèses est celle qu'affiche le catalogue, dans la troisième colonne de l'affichage d'un nom.
Exemple : voir IO.

GETC --- c

Réalise la lecture d'un caractère sur le stockage de masse (cassette ou disquette). GETC laisse toujours un caractère sur la pile, même en cas d'erreur de lecture. Il appartient au programmeur de tester la variable D-ERR pour savoir si l'opération s'est bien passée, et si le caractère lu est correct.
GETC est vectorisé à travers le tableau SYSVEC (position 16) et exécute le mot (KKEY) ou (DKEY) suivant que le stockage de masse est la cassette ou la disquette (voir CASS et DISK).
Avant d'utiliser GETC pour lire un ensemble de données, il faut initialiser l'opération d'entrée-sortie en utilisant IO.

Après utilisation de GETC, il est conseillé de terminer l'opération par : Ø IO
Exemple: Voir IO

IO n ---

Mot système utilisé pour initialiser ou clore une opération d'entrée-sortie sur cassette ou disquette. IO exécute (DIO) ou (KIO) suivant que le périphérique positionné comme stockage de masse est la cassette ou la disquette.
Les valeurs possibles de n sont :

n = Ø Termine une opération de lecture ou d'écriture.

n = 1 Initialise une opération de lecture. Cherche le fichier sur cassette ou disquette, et se place au début.

n = 2 Initialise une opération d'écriture d'un nouveau fichier. Crée un nouveau fichier, avec son nom (voir DNAM et FNAM!).
En mode disquette, création d'une entrée au catalogue.

n = 3 (Seulement en version disquette). Initialise une opération de réécriture d'un fichier existant sur disque.

IO est un mot système, mais vous pouvez l'utiliser pour écrire caractère par caractère sur disquette ou sur cassette.
Exemple : écriture sur stockage de masse de ce que vous tapez au clavier...

```
: ECRIRE 2 IO
        BEGIN KEY PUTC PTRIG UNTIL          ( Ecrire jusqu'à ...)
        Ø IO ;
```

Puis :

```
" TESTIO" FNAM!                              ( Fixe le nom de fichier )
1 FTYPI 255 FASC!                            ( Fichier de données ASCII )
ECRIRE              ( Tapez la phrase de votre choix, appuyez le bouton )
                    ( du crayon optique en même temps que le dernier caractère )
```

Voilà, votre phrase est dans le fichier de nom TESTIO...
Si vous avez des disquettes, vous pouvez vérifier l'existence de ce fichier par l'instruction DIR (voir annexe B).

Pour relire le fichier :

```
: LIR     1 IO
          BEGIN GETC EOF NOT                    ( Lire tant que pas EOF)
          WHILE EMIT REPEAT Ø IO ;
```

Puis : LIRE (Votre phrase doit s'afficher...)

En supposant bien entendu que vous n'avez pas modifié votre nom de fichier depuis (attention, toute utilisation de l'éditeur modifiera le nom placé à l'adresse contenue dans la variable DNAM. Un bon moyen de vérifier le nom est : DNAM @ 11 TYPE

Cela dit, le mot LIRE vous afficherait dans ce cas le dernier écran accédé par l'éditeur... (analogue à LIST. Voir chapitre 11).

IOERR ---

Mot système, affichant le message "IO ERROR" en cas d'erreur dans une lecture ou une écriture sur le stockage de masse.

LOAD n ---

Considère le block n comme flot d'entrée, et l'interprète comme si vous veniez de le taper au clavier. Si le block n'est pas actuellement en mémoire, LOAD le lit sur le stockage de masse.

```
1Ø LOAD
```

Interprète le block 1Ø (écran 1Ø de l'éditeur).

Cette commande peut faire partie de l'écran 5, par exemple... Elle sera alors traitée lors de l'interprétation du block 5. Un moyen de réaliser une application comportant plusieurs écrans (par exemple les écrans 5, 6, 8, 10) est de mettre en fin d'écran une commande LOAD...

```
Fin d'écran 5 :     6 LOAD
Fin d'écran 6 :     8 LOAD
Fin d'écran 8 :    1Ø LOAD
```

En tapant ensuite "5 LOAD", c'est quatre écrans qui seront lus et interprétés.

Une autre manière de procéder est de faire un block "de chargement", en écrivant un écran 1, par exemple les commandes :

```
5 LOAD 6 LOAD 8 LOAD 1Ø LOAD
( ou 1Ø 8 6 5 LOAD LOAD LOAD LOAD)
```

La commande "1 LOAD" provoquera alors l'interprétation des quatre blocks. Voir aussi la commande THRU.

Attention, n doit être non nul !!!... Le numéro de block Ø est réservé

pour indiquer que le flot d'entrée vient du clavier. L'ordre "Ø LOAD" (qui provient par exemple du clavier) n'a donc pas de sens...
En fait, si vous tapez "Ø LOAD", l'action de l'interpréteur sera d'exécuter le contenu du buffer des entrées du clavier (voir TIB). Or, ce contenu est ... "Ø LOAD" !!!... D'où un cercle vicieux, et votre ordinateur reste sans réaction... La seule échappatoire est le bouton de réinitialisation.

MAXBUF --- n

Donne le nombre maximum de buffers qu'on peut fixer, compte tenu de la configuration de votre système (avec ou sans extension mémoire, et compte tenu de l'état actuel du dictionnaire. Voir en annexe l'occupation de la mémoire).
Une fois votre application terminée, vous pouvez minimiser les échanges entre l'unité centrale et le stockage de masse en incorporant dans votre mot final la commmande : MAXBUF BUFFERS

MOTOROFF ---

Arrête le moteur du lecteur-enregistreur de cassettes. Voir MOTORON.

MOTORON ---

Met en marche le moteur du lecteur-enregistreur de cassettes, et fait avancer la bande. Un bon moyen de séparer les enregistrements sur cassette est d'utiliser un mot du genre :

```
: SEPARE (n ---) MOTORON WAIT MOTOROFF ;  Ex :  5ØØ  SEPARE
```

Ou si vous voulez faire un positionnement de la cassette (pour sauter deux ou trois numéros du compteur...) :

```
: POSIT  MOTORON KEY DROP MOTOROFF ;
```

Puis POSIT, et surveillez le compteur de votre lecteur-enregistreur de cassettes, tapez une touche pour arrêter le déroulement de la bande.

OFFSET --- adr

Mot système réservé pour une utilisation future (gestion des disquettes par blocks physiques).

PREV --- adr

Variable contenant l'adresse du buffer disque le plus récemment référencé.
Comme FIRST, PREV donne l'adresse du début de la zone mémoire effectivement occupée par le buffer, deux octets avant le début de la zone utilisable pour le stockage de données (ces deux octets contiennent le numéro du block).

```
10 BLOCK HEX U. 9BDE OK (par exemple)
PREV    U. 9BDC OK
DECIMAL PREV    ? 10 OK
```

Si maintenant on utilise UPDATE (voir ce mot), c'est le block 10 qui sera marqué comme modifié.

PUTC c ---

Réalise l'écriture d'un caractère sur le stockage de masse. L'opération d'écriture doit avoir été initialisée par le mot IO (voir ce mot). Il appartient au programmeur de tester la variable D-ERR pour savoir si l'opération s'est bien déroulée.
Exemple : voir IO.
PUTC est vectorisé à travers le tableau SYSVEC (position 17) et exécute le mot (KEMIT) ou (DEMIT) suivant que le stockage de masse est la cassette ou la disquette.

R/W adr n cond ---

Mot de base pour la lecture ou l'écriture d'un block sur cassette ou sur disquette. Par défaut, R/W appelle le mot (FR/W) (voir ce mot pour la description des paramètres).
Toutefois, R/W est vectorisé à travers le tableau SYSVEC (position 12), de sorte que vous pouvez l'utiliser pour lire ou écrire des blocks sur un stockage de masse non prévu (pourquoi pas un disque dur...).

RS-B n ---

(Abréviation de RS-232-bits). Permet de fixer le nombre de bits pris en compte pour une transmission série RS-232 par le port de communication. Les valeurs possibles de n sont 7 ou 8 (7 bits ou 8 bits). Voir aussi RS-S.

RS-S n ---

(Pour RS-232-speed). Fixe la vitesse de transmission série RS-232 par le port de comunication.
Les valeurs possibles de n sont :

n = 1	11Ø bauds	n = 5	24ØØ bauds
n = 2	3ØØ bauds	n = 6	48ØØ bauds
n = 3	6ØØ bauds	n = 7	96ØØ bauds
n = 4	12ØØ bauds		

Après avoir fixé le nombre de bits transmis (par RS-B) et la vitesse de la communication (par RS-S), on peut utiliser (PIO) pour transmettre des caractères.

SAVE-BUFFERS ---

Sauvegarde sur le stockage de masse le (ou les) block(s) marqué(s) comme modifiés (voir UPDATE). SAVE-BUFFERS agit différemment suivant que le stockage de masse est la cassette ou la disquette.
En mode "cassette", SAVE-BUFFERS sauvegarde uniquement le dernier block référencé, s'il a été modifié.
En mode "disquette", SAVE-BUFFERS sauvegarde tous les blocks qui sont marqués comme modifiés.
SAVE-BUFFERS est vectorisé à travers le tableau SYSVEC, et exécute (FLUSH) ou (KFLUSH) suivant le stockage de masse sélectionné (voir CASS et DISK). Il vous est aussi possible d'écrire votre routine de sauvegarde, et de mettre son cfa en position 14 du tableau SYSVEC. C'est elle que SAVE-BUFFERS appellera.

SKIP ---

En mode "cassette", SKIP permet de "sauter" le prochain fichier de la cassette, quel que soit son type (écran de l'éditeur, sauvegarde-dictionnaire, programme BASIC, fichier de données, etc.). L'action exacte de SKIP est de chercher le début du prochain fichier, d'écrire son nom à l'écran, et de le lire jusqu'au bout (caractère par caractère) avant d'arrêter le lecteur-enregistreur de cassettes.

SKIPB n ---

En mode "cassette", SKIPB positionne la bande immédiatement après le block n. Son action exacte est de rechercher un fichier dont le nom est "BLOCKn.DAT", de le lire caractère par caractère (en utilisant SKIP), et d'arrêter le lecteur-enregistreur de cassettes immédiatement après. Au passage, SKIPB affiche les noms des fichiers rencontrés. Si la cassette ne contient aucun fichier correspondant, la cassette se déroule jusqu'au bout.
SKIPB peut donc servir à vérifier le contenu d'une cassette, il suffit de donner un numéro de block inexistant, de mettre la cassette au début, et de s'armer de beaucoup de patience...
Si on connaît les numéros des blocks de la cassette, SKIPB permet aussi de se positionner après le dernier fichier. Une pratique raisonnable est d'utiliser des numéros de block consécutifs, et de ne retenir que le dernier. Il est aussi prudent de sauvegarder deux fois.

THRU n1 n2 ---

Interprète l'un après l'autre les blocks (ou écrans de l'éditeur) dont les numéros vont de n1 à n2 (n2 doit être supérieur à n1).

```
12 16 THRU équivaut à : 12 LOAD 13 LOAD 14 LOAD 15 LOAD 16 LOAD
```

Il est même possible de commencer une définition dans un écran, et de la terminer dans l'écran suivant...

UPDATE ---

Marque comme modifié le block le plus récemment référencé (celui dont l'adresse est dans la variable PREV).
En mode "cassette" (voir CASS), le prochain SAVE-BUFFERS recopiera ce block sur la cassette (sauf si vous avez référencé un autre block entretemps), ainsi que la commande CNT-W de l'éditeur.
En mode "disquette" (voir DISK), ce block sera automatiquement recopié sur disque dès que nécessaire... Cela peut se produire parce que vous avez tapé SAVE-BUFFERS (ou CNT-W sous l'éditeur), mais aussi parce que le système a besoin de récupérer la place mémoire occupée par le buffer du block concerné (en général pour allocation d'un nouveau block).

USE --- adr

Variable système contenant l'adresse du buffer à utiliser lors du prochain accès au stockage de masse. Ce buffer est celui qui est considéré comme le plus récemment référencé.
USE est surtout employé par le système, lors de l'attribution d'un nouveau buffer à un block. Voir BLOCK et BUFFER.

Chapitre 11

Utilitaires et mots divers

Ce chapitre regroupe des mots difficilement classables (appels de sous-programmes en langage machine, gestion des messages d'erreur...), des commodités d'écriture (commentaires dans un source FORTH, affichage des piles, affichage d'un écran de l'éditeur...), nécessaires pour le standard (79-STANDARD), et des mots spécifiques, rendus nécessaires par l'implantation en ROM (SYSVEC), ou utilisant des particularités de votre ordinateur (AUTORUN).

(---

Utilisé sous la forme : (ccccccccccccccccccccc)
Permet l'insertion de commentaires dans une définition. Les caractères entre les parenthèses sont ignorés du compilateur et de l'interpréteur. La parenthèse ouvrante est un mot FORTH, donc doit être précédée et suivie d'un espace. La parenthèse fermante est un terminateur, et n'est pas nécessairement précédée d'un espace. (Par contre le caractère suivant ne peut être qu'un espace, pour isoler le mot suivant).
Si vous voulez qu'une partie d'un écran de l'éditeur ne soit pas interprétée, il suffit de la mettre entre parenthèses (attention aux blancs).

Dans ce cas, il vous faudra vous souvenir qu'un texte entre parenthèses ne peut excéder 255 caractères. Si nécessaire, utilisez plusieurs parenthèses...
Une autre méthode est d'utiliser le mot ;S (voir chapitre 4).

(ABORT) ---

Mot système appelé par ABORT. Son rôle est de réinitialiser la pile des données, les paramètres de l'écran (sauf les couleurs), le vocabulaire courant (redevient FORTH s'il ne l'était pas), avant d'appeler le mot QUIT pour rendre la main à l'interpréteur.

(LINE) n1 n2 --- str

Donne les paramètres (adresse et longueur) de la chaîne de caractères qui constitue la ligne n1 du block n2. La longueur est toujours 40.
Les lignes du block sont numérotées de 0 à 24.
Pour une gestion de fichier utilisant les blocks, il est commode de repérer les enregistrements par le numéro de block, le numéro de la ligne, et le décalage par rapport au début de la ligne du début de l'enregistrement. Dans la mesure du possible, on peut même essayer de faire commencer les enregistrements en début de ligne (longueur multiple de 40).

.LINE n1 n2 ---

Affiche la ligne n1 du block n2. Comme pour (LINE), les lignes sont numérotées de 0 à 24.
Il est préférable pour votre tranquillité que cette ligne ne contienne que des caractères ASCII affichables (pas trop de caractères de contrôle...).

```
: TITRES( n1 n2 --)
        1+ SWAP DO 0 I .LINE LOOP ;
12 17 TITRES        ( Affiche la première ligne des écrans 12 à 17 de l'éditeur)
```

A vous d'écrire un titre en première ligne de chaque écran...

.S ---

Affichage non destructif du contenu des deux piles (pile des données, et pile des retours).
Utilisation intéressante pour l'initiation, mais aussi pour la mise au point (debugging).

```
1 2 3 4 .S  ( Voir la pile des données: PARAMETER STACK:...)
SWAP .S     ( Effet de SWAP sur la pile...)
+ .S        ( Effet de l'addition ...)
```

Pour la pile des retours (RETURN STACK), visualisons par exemple le fonctionnement de la boucle DO...LOOP :

```
: TEST .S DO I .S LOOP .S ; 5 2 TEST
```

On retrouve la valeur de l'indice sur la pile des retours, ainsi que la valeur limite (5). Le premier .S montre l'effet de DO (empile 5 et 2), et le dernier .S montre l'effet de LOOP lorsque l'indice est supérieur à la limite.
Essayez d'autres valeurs... 11 Ø TEST... -2 -5 TEST...
En débugging, l'insertion provisoire de .S dans vos mots permet de vérifier leur action sur les piles.

79-STANDARD ---

Ce mot est indispensable pour que le FORTH du TO-7 soit standard, mais le standard FORTH lui-même ne décrit pas l'action qu'il doit réaliser...
Sur le TO-7, ce mot fixe la longueur des noms du dictionnaire à 31 caractères (voir le mot WIDTH)

?ERROR cond n ---

Réalise l'affichage du message d'erreur numéro n si la condition cond est "vrai". Dans ce cas, le message est précédé du dernier mot analysé dans le flot d'entrée, et après l'affichage ?ERROR appelle le mot ABORT pour nettoyer la pile et rendre la main au clavier.

```
1 7 ?ERROR ?ERROR ?FS ERROR
```

Les messages d'erreur sont regroupés dans le tableau ERR. Le message d'erreur numéro 7 est le message "FS" (full stack, pile pleine).

ABORT ---

Par défaut, ABORT exécute le mot (ABORT). C'est à dire qu'il réinitialise la pile des données et la pile des retours, quitte éventuellement le mode compilation, et rend la main au clavier. Il est habituellement employé pour terminer un traitement d'erreur.
En fait, ABORT exécute le mot dont le cfa est contenu dans le tableau SYSVEC en position 5, de sorte qu'on peut modifier son action. Par exemple, on peut se servir de cette possibilité pour faire un traitement spécifique des erreurs (voir ONERR et l'annexe D).

AUTORUN ---

Utilisé sous la forme AUTORUN XX avant de sauvegarder un dictionnaire FORTH (voir SAVE). Lorsque par la suite on rechargera ce dictionnaire (par RUN), le mot XX s'exécutera automatiquement. Ce mot est surtout intéressant lorsque votre application est achevée, et que vous voulez la faire exécuter par des personnes qui ne connaissent pas encore FORTH... Dans ce cas, pensez aussi à employer "Ø SAVE" pour que l'utilisateur de votre application n'ait qu'à choisir l'option "Un programme enregistré".
AUTORUN peut aussi être utile si vous avez par exemple une imprimante qui demande à être initialisée lors de la mise en marche du système, ou si vous voulez positionner vos couleurs préférées de manière automatique à la mise sous tension.

```
: ZOU ( ---) BLACK FRAME GLOBAL WHITE COLOR LOCAL EMPTY-BUFFERS ;
     AUTORUN ZOU 2 SAVE
```

Lorsque vous ferez "2 RUN", les couleurs se mettront en place...

BELL ---

Fait un bip. En fait, BELL joue l'accord do-mi-sol-do, en durée 2 (voir DUR, chapitre 7). Puisque c'est un mot musical, il est sensible aux paramètres musicaux (sauf la durée).

```
: BIPS ( ---) 6 1 DO I OCT BELL LOOP 4 OCT ; BIPS ...
```

C/L **--- adr**

Variable utilisée par VLIST et par l'éditeur, contenant le nombre de caractères maximum par ligne pour les affichages ultérieurs. Si cette valeur doit être dépassée lors de l'affichage du mot suivant, VLIST passe à la ligne.
D'où par exemple : `2 C/L ! VLIST 40 C/L !`
pour un VLIST en une colonne...
Si l'imprimante le permet, on peut aussi obtenir un VLIST en 80 colonnes.

CALL **adr ---**

Appelle le sous-programme en langage machine à l'adresse adr. Les registres sont sauvegardés avant l'appel, et restaurés au retour.
Si vous voulez passer des paramètres au sous-programme, il vous suffit de les ranger avant l'appel dans les variables REGD, REGX, REGY et REGU.

COLD **---**

Provoque un redémarrage à froid de votre système, dans les mêmes conditions que lors de la mise sous tension. En particulier, si vous avez allumé votre lecteur de disquettes après le démarrage du FORTH, ne comptez pas sur COLD pour charger le système d'exploitation. Mais si vous n'avez écrit que des essais sans importance, COLD réinitialise le dictionnaire, les buffers, les piles, et affiche le copyright.

EDIT **n ---**

Entre en mode édition sur l'écran numéro n. Voir l'annexe A pour les commandes de l'éditeur.

ERR --- adr

Table de caractères ASCII contenant les messages d'erreur. Chaque message est composé d'une suite de caractères ASCII, précédée par un octet de longueur.
Dans la cartouche ROM, 19 messages (numérotés Ø à 18) sont prévus. Vous pouvez les afficher en utilisant le mot MESSAGE. Pour des raisons d'occupation mémoire, ils ne comportent que 2 caractères. Si vous les trouvez trop laconiques, il vous est possible de réécrire (en RAM) un tableau des messages d'erreur, et de mettre son adresse dans le tableau SYSVEC en position 18 (voir annexe D). FORTH utilisera alors vos messages à la place des siens, et vous ne serez pas limités par la longueur, ni par la langue (vous pouvez les écrire en français)...
Exemple :

```
CREATE  ERR1  300  ALLOT  ( Réserver la place, pour pouvoir oublier la suite)
VARIABLE ADRESSE                    ( Adresse où ranger le prochain message)
' ERR1 ADRESSE !                          ( Au début, pfa de ERR1)
: MESSAGE!  ( str --- range < str > à la fin de ERR1)
        1+  >R  1-  ADRESSE  R@ CMOVE  R>  ADRESSE  +! ;
                                          ( Un peu de technique...)
```

Puis :

```
" Je n'ai pas l'honneur de connaître..." MESSAGE!
" Ma pile est vide..." MESSAGE!
" Mon dictionnaire est plein !!!" MESSAGE!
```

Et ainsi de suite... La longueur totale des messages (compter un octet de plus par message, l'octet de longueur) ne doit pas dépasser 3ØØ. Vous pouvez terminer par FORGET ADRESSE pour oublier les mots inutiles...
Et enfin :

```
' ERR1  CFA  18  SYSVEC !                ( cfa de ERR1 dans SYSVEC)
```

Dorénavant, votre ordinateur ne vous répondra plus "ES ERROR", mais

```
" Ma pile est vide... ERROR".
```

ERROR n1 --- n2 n3

C'est le mot qui est appelé par le mot ?ERROR s'il doit effectivement faire un traitement d'erreur.
ERROR affiche le dernier mot du flot d'entrée, puis " ?", puis le

message d'erreur numéro n1. Ensuite, il réinitialise la pile et appelle le mot ABORT pour rendre le contrôle au terminal en mode exécution. ERROR laisse sous la pile l'offset du prochain caractère à lire dans le flot d'entrée (n2), ainsi que la valeur n3 qui est le numéro du block qui constitue le flot d'entrée (Ø pour le clavier). Ces valeurs peuvent être récupérées pour une utilisation ultérieure, par exemple pour un traitement des erreurs. Par exemple, le mot WHERE utilise ces valeurs pour réafficher l'écran qui contenait une erreur, et pour placer le curseur après cette erreur.

GO ---

Mot système utilisé pour le démarrage du système. Son action est de réinitialiser les buffers, puis d'exécuter le mot dont le cfa est en position 6 dans le tableau SYSVEC. Par défaut, ce mot affiche le copyright et rend la main au clavier.
L'action de AUTORUN XX est simplement de fixer cette valeur, de sorte qu'au démarrage ce soit XX qui s'exécute...

```
: XX BELL BELL BLACK FRAME ; AUTORUN XX GO
```

L/SCR --- adr

Variable utilisée par l'éditeur pour mémoriser le nombre de lignes de l'écran. Sa valeur par défaut est 25, mais vous pouvez choisir toute autre valeur entre 1 et 25.
Si par exemple vous voulez faire des essais avec l'éditeur, et visualiser simultanément l'exécution, vous pouvez définir :

```
1Ø L/SCR ! : VISU 1Ø 24 WINDOW HOME ; 1Ø EDIT
```

Il suffit alors de placer le mot WINDOW dans votre écran. Lorsque vous aurez terminé votre entrée, placez vous sur la dernière ligne de l'écran pour taper CNT-Y. Ce que vous taperez ensuite en mode interactif sera confiné en bas de l'écran du téléviseur, et ne cachera pas le texte que vous avez entré sous l'éditeur. Mais bien entendu, votre block 1Ø ne sera rempli qu'aux deux-cinquièmes... A vous de choisir entre le confort et la place mémoire...

LIST n ---

Affiche le contenu du block numéro n sur le périphérique de sortie. Le block en question ne doit bien entendu contenir que des caractères affichables. LIST peut servir à visualiser le contenu d'un écran de l'éditeur, ou un fichier ASCII sauvegardé sous forme de blocks. Vous pouvez à tout moment arrêter (ou faire continuer) le défilement en tapant une touche quelconque. Lorsque le défilement est interrompu, la touche STOP arrête l'exécution du mot, comme pour VLIST (voir ?STOP, au chapitre 9).

```
: LISTING  ( n1 n2 ---)
        1+ SWAP PRINTER DO I LIST LOOP CR CONSOLE ;
10 15 LISTING  ( Imprime les blocks 10 à 15)
```

MESSAGE n ---

Réalise effectivement l'affichage du message d'erreur numéro n. MESSAGE est appelé par ERROR.
Vous pouvez l'utiliser si vous voulez afficher des messages d'erreur, sans pour cela revenir au clavier (traitement personnalisé des erreurs).

```
4 MESSAGE NU OK
```

Cette possibilité est surtout intéressante si vous avez choisi vos propres messages d'erreur (voir ERR).

ONERR ---

Utilisé sous la forme ONERR XX pour positionner XX comme étant le mot à exécuter en cas d'erreur. Par défaut, c'est le mot (ABORT) qui est sélectionné.
En le remplaçant par XX, vous pouvez donc personnaliser votre traitement d'erreur.

```
: GAFFE!! BELL BELL BELL RED FRAME
          KEY DROP CYAN FRAME (ABORT) ;
```

ONERR GAFFE!! Maintenant, votre attention sera attirée en cas d'erreur... Tapez par exemple un point pour provoquer une erreur "pile vide"...
En fait, l'action de ONERR est de positionner le mot appelé par ABORT (voir ce mot).

QUIT ---

C'est lui qui réalise la boucle sans fin de l'interpréteur, qui consiste à lire le flot d'entrée et à le traiter. C'est en quelque sorte le "superviseur" du langage...
Lorsqu'on l'appelle explicitement, QUIT initialise la pile des retours et fixe le flot d'entrée au clavier.

REDIT ---

Réédite le dernier écran utilisé par l'éditeur. Ce petit mot est bien utile lorsque vous ne savez plus trop avec quel écran vous étiez en train de travailler...
Dans cette situation, utilisez plutôt REDIT pour rééditer le dernier écran référencé.

REGD --- **adr**
REGU --- **adr**
REGX --- **adr**
REGY --- **adr**

Variables servant à passer des paramètres à un sous-programme en langage machine appelé par le mot CALL.
Lors de l'appel, CALL sauvegarde (sur la pile système) les registres du 68Ø9, et les charge avec les valeurs contenues dans REGD, REGU, REGX et REGY. Au retour du sous-programme, les valeurs des registres lors de l'appel sont restaurées.

RND **n1 --- n2**

Donne un nombre aléatoire. n2 est un nombre entier "au hasard", entre Ø et n1 – 1. Il s'agit en fait d'un nombre pseudo-aléatoire, calculé à partir de la valeur de deux octets dont le contenu est modifié par chaque opération de lecture du clavier ou d'affichage d'un caractère (KEY et EMIT incrémentent la valeur qui sert de point de départ à la génération du nombre n2).
Exemple : simulation du jet d'un dé à jouer

```
: DE 6 RND 1+ ;          ( 6 RND donne un nombre entre Ø et 5)
                         ( 1+ a pour effet de donner un résultat entre 1 et 6)
```

Ou encore : musique aléatoire

```
: OCT-ALEA  5 RND 1+ OCT ;
: DUR-ALEA  30 RND 2+ DUR ;
: NOTE-ALEA OCT-ALEA DUR-ALEA 12 RND 49 + NOTE ;
: ZIK-ALEA  BEGIN NOTE-ALEA PTRIG UNTIL ;
```

Vous trouverez certainement beaucoup d'autres utilisations de ce mot.

RUN n --- adr

Lit sur le périphérique d'entrée le dictionnaire sauvegardé avec le numéro n (voir SAVE), le charge en mémoire à la place du dictionnaire existant, puis exécute le mot GO. L'effet apparent est l'exécution du mot fixé par AUTORUN. Ce mot doit entre autres réaliser un EMPTY-BUFFERS.
Le précédent dictionnaire en RAM est perdu.
RUN laisse sur la pile l'adresse du premier octet disponible à la fin du dictionnaire chargé.
Si vous disposez d'un TO-7 ou d'un TO7-70, l'option "un programme enregistré" (option numéro 2) lors de la mise sous tension consiste pour FORTH à chercher sur la disquette un dictionnaire portant le numéro Ø, et à le charger si possible.

SAVE n ---

Sauvegarde sur cassette ou sur disquette l'état actuel du dictionnaire en RAM, avec le numéro n. Ce dictionnaire peut être rechargé par RUN (voir ce mot).
Si vous tapez "Ø SAVE", vous pourrez (avec un TO7 ou un TO7-70) utiliser à la mise sous tension l'option "2 - Un programme enregistré" pour charger automatiquement votre dictionnaire sans employer RUN.
Exemple : en fin d'application...

```
AUTORUN MENU Ø SAVE
```

En fait, SAVE sauvegarde un fichier dont le nom est du type "DICTxxx.FTH". Ce fichier contient le dictionnaire RAM, mais aussi la zone système (de la page zéro du FORTH à la fin du dictionnaire. Voir annexe E).

La page zéro du moniteur n'est pas sauvegardée, de sorte que les paramètres musicaux, les couleurs, etc. ne sont pas sauvegardés. Votre application devra donc comporter une initialisation.
Une même disquette peut comporter plusieurs applications compilées.

SCR --- adr

Variable contenant le numéro du dernier écran accédé par l'éditeur... REDIT utilise SCR. il est défini par :

```
: REDIT SCR @ EDIT ;
```

SYSVEC n ---

Tableau système, contenant les cfa de routines que l'utilisateur peut modifier.
En réalité, SYSVEC n'est pas un tableau au sens habituel du FORTH. Lorsqu'on tape par exemple "2 SYSVEC", on obtient sur la pile une adresse de la page Ø du FORTH, et le nombre 16 bits à cette adresse est le cfa du mot actuellement employé pour la lecture d'un caractère au clavier (appelé par KEY). De sorte qu'en changeant cette valeur (qui est en RAM), on modifie l'action du mot KEY.
Voir l'annexe D pour une étude plus précise de cette question.

WAIT n ---

Attend approximativement n millisecondes. Le mot WAIT peut être utilisé pour réaliser des temporisations.

```
: METRONOME  ( n --- fait un bip tous les n dixièmes de seconde)
        BEGIN DUP 1ØØ * WAIT 7 EMIT PTRIG
UNTIL ;
```

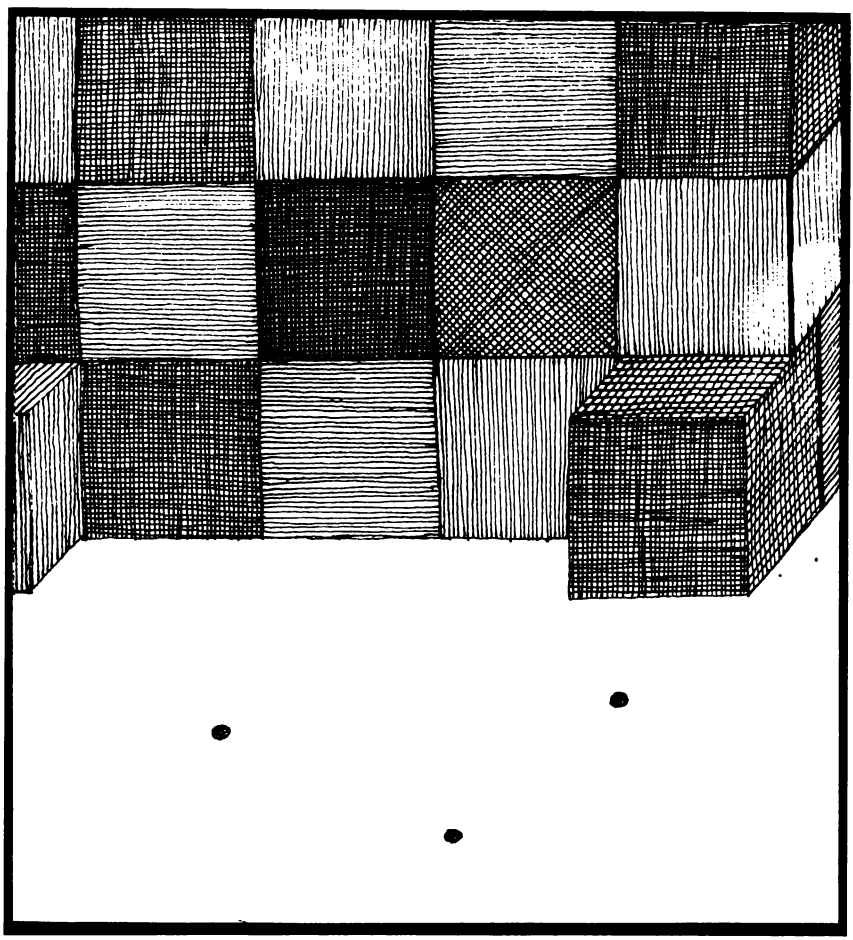
WHERE ---

Si FORTH a rencontré une erreur en interprétant un écran de l'éditeur (lors d'un LOAD, THRU, ou d'un CNT-Y sous l'éditeur), WHERE réalise le retour en édition dans l'écran concerné, et le curseur est positionné sur le début du mot qui suit le mot qui a provoqué l'erreur.

N'utilisez pas WHERE si vous n'avez pas eu de message d'erreur, il interprèterait les nombres sur la pile comme étant un numéro de block et un nombre d'octets dans le block. Il pourrait très bien vous créer un nouveau block sur disque...

Deuxième partie

Compléments et annexes



Annexe A

L'éditeur

A.1 Généralités
Le fonctionnement de l'éditeur

L'éditeur constitue un vocabulaire qu'on peut sélectionner par le mot EDITOR.
Cette sélection est réalisée automatiquement par les mots EDIT, REDIT, ou WHERE (voir chapitre 11). Ces mots appellent en outre le mot EDSCR, qui lit les touches tapées et exécute des commandes, en fonction de la touche.
Ces commandes agissent de la manière suivante :
Le principe est que l'écran du téléviseur reflète à tout moment le contenu du block en cours d'édition. A l'entrée dans l'éditeur, le numéro de ce block est sauvegardé dans la variable SCR. Initialement, ce block est à blanc, c'est à dire qu'il ne contient que des caractères "espace".
Si la touche tapée correspond à un caractère affichable (code ASCII supérieur ou égal à 32), ce caractère est affiché à l'écran, et remplace en mémoire (dans le block édité) le caractère qui correspond à la position du curseur à l'écran. En outre, le block est alors marqué comme modifié.

Si la touche tapée ne correspond pas à un caractère affichable, c'est une touche de contrôle (code ASCII inférieur ou égal à 31). Dans ce cas, la commande correspondante s'exécute. Elle peut consister en un déplacement du curseur, une insertion ou suppression de ligne ou de caractère, une commande de lecture ou d'écriture sur un périphérique, etc.

L'éditeur comporte donc en fait 32 commandes, regroupées dans une table CMD. Cette table est vectorisée à travers le tableau SYSVEC (position 19 : voir annexes A-3 et D). Il vous est donc possible de réécrire les commandes de l'éditeur si celles de la cartouche ROM ne vous conviennent pas.

A.2 Mise en œuvre

A l'entrée dans l'éditeur par "n EDIT" (ou REDIT ou WHERE), le block n s'affiche à l'écran.

Dans le cas où ce block n'est pas actuellement dans un buffer :
- Si vous travaillez avec des disquettes, il est lu automatiquement sur la disquette.
- Si vous travaillez sans stockage de masse, ou avec un lecteur-enregistreur de cassettes, la commande réserve un buffer, et le met à blanc (rempli d'espaces). Pour lire votre block sur cassette, il faut taper CNT-R (voir les commandes ci-dessous). Attention, si vos buffers sont tous occupés, les blocks ne seront pas sauvegardés automatiquement lors de l'allocation d'un nouveau buffer.

Pour bien comprendre ce qui se passe, il suffit de noter qu'en écriture la sauvegarde est réalisée par SAVE-BUFFERS, et qu'en lecture du stockage de masse, c'est BLOCK ou BBLOCK qui s'exécute suivant qu'on dispose ou non de disquettes. Consultez le chapitre 10 pour le fonctionnement de ces mots.

Le block souhaité étant maintenant en mémoire dans un buffer (à blanc ou non), les commandes disponibles sont les suivantes:

Touches particulières

ENTREE
Place le curseur en début de ligne suivante.

RAZ
Remise à blanc de l'écran. ATTENTION, cette commande est dangereuse.

STOP
Quitte le mode édition sans sauvegarde ni compilation.

EFF
Efface le caractère sous le curseur. Décalage à gauche de la fin de ligne.

INS
Insère un espace à la position du curseur. Décalage à droite de la fin de la ligne.

FLECHES DE DIRECTION [↑] [←] [→] [↓]
Déplacements du curseur d'une case dans la direction.

FLECHE EN U [↵]
Place le curseur alternativemant en début de première et de dernière ligne de l'écran.

ACC
Comme hors de l'éditeur, fait partie d'une séquence de trois touches, pour entrer un caractère minuscule accentué. Cette touche ne correspond pas à une commande de l'éditeur.

CNT-A
Remplace le caractère sous le curseur par un [.

CNT-B
Equivaut à STOP.

CNT-C
Copie la ligne du curseur dans le tampon d'édition. Cette ligne peut être recopiée à l'écran par CNT-E. Si le tampon est plein, pas d'action (l'ordinateur prévient par un bip).

CNT-D
Place le curseur alternativement en début-fin de ligne.

CNT-E
Recopie à la place de la ligne du curseur la dernière ligne du tampon d'édition. L'ancienne ligne du curseur et les suivantes sont décalées vers le bas. Si l'écran est plein (dernière ligne non vide), pas d'action (l'ordinateur prévient par un bip).

CNT-F
Tabulation à gauche. Les positions préétablies vont de cinq en cinq caractères à partir du début de la ligne (positions 1, 6, 11, etc.).

CNT-G
Tabulation à droite.

CNT-H
Equivaut à la flèche à gauche.

CNT-I
Equivaut à la flèche à droite.

CNT-J
Equivaut à la flèche vers le bas.

CNT-K
Equivaut à la flèche vers le haut.

CNT-L
Equivaut à la touche RAZ. ATTENTION, commande dangereuse.

CNT-M
Equivaut à la touche ENTREE.

CNT-N
Sélectionne ou désélectionne l'affichage du numéro de l'écran en cours d'édition. Ce numéro peut cacher des caractères, et vous pouvez écrire dessus sans problème. Ce n'est qu'un artifice d'affichage, qui ne fait pas partie du texte de l'écran.

CNT-O
Passe en mode édition sur l'écran précédent. Le tampon d'édition n'est pas réinitialisé, de sorte que cette commande peut être utilisée avec CNT-C (ou CNT-X) et CNT-E pour recopier ou déplacer des lignes d'un écran à l'autre.

CNT-P
Recopie le contenu de l'écran sur l'imprimante. Cette commande n'est bien sûr utilisable que si vous avez une imprimante.

CNT-Q
Passe en mode édition sur l'écran suivant. Le fonctionnement est analogue à celui de CNT-O.

CNT-R
Relit le block en cours d'édition depuis le stockage de masse. Cette commande permet de relire l'original sans valider les changements effectués. En mode "cassette", elle est nécessaire pour réaliser la lecture effective depuis le stockage de masse lors de l'entrée dans l'éditeur (ou après CNT-O ou CNT-Q), si le block concerné n'est pas déjà en mémoire.

CNT-S
Scinde en deux la ligne, à la position du curseur. Les lignes suivantes sont décalées vers le bas, et la fin de la ligne en cours est déplacée au début de la ligne suivante. Si l'écran est plein (dernière ligne occupée), pas d'action (l'ordinateur prévient par un bip).
CNT-S sert aussi à insérer des lignes. Il suffit de se mettre en début de ligne avant d'exécuter la commande.

CNT-T
Réinitialise le tampon d'édition. Ce tampon occupe toute la place disponible en RAM qui n'a pas été réservée pour des buffers. Sa taille approximative en k-octets est donnée (hors éditeur) par: MAXBUF – #BUF.

CNT-U
Réunit deux lignes consécutives. La ligne qui suit la ligne du curseur est recopiée à partir du curseur, et les lignes suivantes sont décalées vers le haut. La fin de l'ancienne ligne du curseur est perdue. S'il n'y a pas assez de place en fin de ligne pour recopier la ligne suivante, CNT-U est inopérante (l'ordinateur émet un bip pour attirer votre attention).

CNT-V
Equivaut à la touche ACC.

CNT-W
Réalise un SAVE-BUFFERS. L'effet en version cassette est de sauvegarder l'écran en cours s'il a été modifié depuis sa lecture. En version disquette, CNT-W sauvegarde tous les écrans modifiés. Si la commande est inopérante, c'est que votre écran n'est pas marqué comme modifié. Pour forcer la sauvegarde (par exemple pour sauvegarder plusieurs fois sur cassette), il suffit d'activer toute commande d'écriture. Le plus simple est de retaper le caractère sous le curseur, puis de retaper CNT-W.

CNT-X
Supprime la ligne où se trouve le curseur. Les lignes suivantes sont décalées vers le haut... La ligne supprimée est recopiée dans le tampon d'édition, de sorte que l'utilisation conjointe de CNT-X et de CNT-E permet de déplacer des lignes dans un écran, ou d'un écran à l'autre (le tout est de ne pas sortir de l'éditeur, et de ne pas réinitialiser le tampon par CNT-T).

CNT-Y
Quitte l'éditeur et compile l'écran en cours, comme le ferait la commande STOP suivie de la commande "n LOAD".

CNT-Z
Remplace le caractère sous le curseur par].

Toutes les autres touches fonctionnent comme d'habitude, c'est-à-dire qu'elles remplacent le caractère sous le curseur par le caractère tapé. Les minuscules et les caractères accentués sont autorisés (attention ! N'utilisez pas de caractère accentué dans un nom de mot FORTH).

A.3 La table des commandes Les mots de l'éditeur

Fonctionnement de EDSCR

Toute touche tapée en mode édition est mémorisée dans la variable LK (Last Key). Son code ASCII est ensuite interprété comme un numéro de commande, après avoir été limité à 31. La commande 31 a pour action d'écrire le caractère à l'écran et dans le buffer du block en cours d'édition.
La table des commandes est composée comme suit :

Code ASCII (numéro de la commande)	Touche du clavier	Commande	
Ø	Inaccessible	BELL	(pour remplissage)
1	CNT-A (caractère "["..)	BELL	(pour remplissage)
2	STOP ou CNT-B	BYE	(fin d'édition)
3	CNT-C	LCOP	(copie la ligne dans le tampon)
4	CNT-D	< C >	(curseur début-fin de ligne)
5	CNT-E	LN >	(rappelle la dernière ligne du tampon)
6	CNT-F	< T	(tabulation à gauche)
7	CNT-G	T >	(tabulation à droite)
8	CNT-H/flèche à gauche	H –	(déplacement curseur à gauche)
9	CNT-I/flèche à droite	H +	(déplacement curseur à droite)

10	CNT-J/flèche vers le bas	V +	(déplacement curseur vers le bas)
11	CNT-K/flèche vers le haut	V –	(déplacement curseur vers le haut)
12	CNT-L/touche RAZ	RAZ	(remise à blanc du buffer)
13	CNT-M/touche ENTREE	CCR	(passage en début de ligne suivante)
14	CNT-N	NB	(affichage du numéro ou non)
15	CNT-O	B –	(édition du block précédent)
16	CNT-P	HCOP	(impression du buffer)
17	CNT-Q	B +	(édition du block suivant)
18	CNT-R	RNEW	(relecture sur stockage de masse)
19	CNT-S	SPLIT	(scinde la ligne en deux)
20	CNT-T	ZTP	(réinitialisation du tampon)
21	CNT-U	GLUE	(réunion de deux lignes consécutives)
22	CNT-V ou touche ACC	BELL	(pour remplissage)
23	CNT-W	SAVE-BUFFERS	(sauvegarde si modifié)
24	CNT-X	LDEL	(suppression de la ligne)
25	CNT-Y	BLOAD	(interprétation et fin d'édition)
26	CNT-Z (donne].code 93)	BELL	(pour remplissage)
27	Inaccessible	BELL	(pour remplissage)
28	INS	INSCH	(insertion d'un espace)
29	EFF	DELCH	(suppression d'un caractère)
30	flèche en U	CUD	(curseur en début-fin d'écran)
31	Toute autre touche...	WCH	(écrit le caractère...)

Commentaires :

Les autres mots de l'éditeur sont des mots plus techniques qui n'interviennent que pour définir les mots de la table.

La zone tampon où sont empilées les lignes supprimées par CNT-X ou copiées par CNT-C commence à la fin de la zone PAD, et s'étend jusqu'au début du premier buffer (FIRST). Si vous n'avez pas sélectionné le nombre maximum de buffers, elle peut donc contenir un ou plusieurs blocks.

Mini-édition : les dimensions de l'écran qui sert à l'éditeur sont contenues dans les variables C/L (nombre de caractères par ligne) et L/SCR (nombre de lignes par écran). Leurs valeurs par défaut sont respectivement 40 et 25 (écran maximum).
En modifiant ces variables avant d'entrer dans l'éditeur, vous pouvez n'utiliser qu'une partie du buffer du block édité, mais garder une zone écran libre pour des messages ou simplement parce que vous préférez comme cela... Vous n'aurez pas une utilisation optimale des buffers, mais cela peut présenter un intérêt si par exemple vous modifiez des commandes...
Si vous éditez en format réduit un écran qui avait été écrit en format standard (25x40), vous n'afficherez pas tout le buffer.

Un exemple de personnalisation : vous trouvez la commande RAZ trop dangereuse, et vous décidez de la remplacer par une commande de recherche d'un mot...
— Sélectionnez le vocabulaire EDITOR comme "courant" (voir CURRENT, et annexe C) :

```
EDITOR DEFINITIONS     ( les prochaines définitions prendront)
                       ( place dans le vocabulaire "EDITOR")
```

— Ecrivez un mot RECHERCHE qui réalise l'action souhaitée...
Définissez une table CMD1 :

```
TABLE  CMD1  32  2*  ALLOT
```

Recopiez CMD dans CMD1 par

```
MOVE ou CMOVE... ( Voir ces mots pour exemples )
```

Modifiez la douzième commande :

```
'  RECHERCHE  CFA  '  CMD1  12  2*  +  !
```

Sélectionnez CMD1 comme nouvelle table de commandes de l'éditeur :

```
FIND  CMD1  19  SYSVEC  !
```

Quittez le vocabulaire EDITOR :

```
FORTH  DEFINITIONS
```

Cet exemple est du même esprit que celui qui est proposé pour le mot ERR (chapitre 11), à cela près qu'ici on ne réécrit pas toute la table.

Annexe B

Les mots de la disquette système

Principes généraux

Pour communiquer avec les lecteurs de disquettes, FORTH doit être capable d'accomplir certaines tâches fondamentales comme positionner la tête de lecture, mettre en marche les moteurs du lecteur, chercher un fichier, etc.
Ces actions sont en général transparentes pour l'utilisateur, et sont gérées par ce qu'on appelle le système d'exploitation. Il s'agit d'un ensemble de programmes en langage machine, qui se trouvent sur chaque disquette-système.

Pour charger en mémoire le système d'exploitation, procéder de la manière suivante :
Avec un TO7 ou un TO7-70 :
Allumer le lecteur de disquettes avant l'unité centrale. L'ordinateur reconnait automatiquement la présence d'un lecteur de disquettes. Lorsque vous taperez l'option "1" (choisir FORTH), il essaiera de lire la disquette. Il vous suffit donc de mettre une disquette système dans le lecteur avant de sélectionner l'option "1". Votre TO7 lit alors la disquette et charge le système d'exploitation.

Une autre possibilité est de sélectionner l'option "2" ("Un programme enregistré", représentée à l'écran par le dessin d'une disquette sur un TO7-70). Il est nécessaire pour cela que la disquette contienne un fichier nommé "DICTØ FTH". Ce fichier contient une sauvegarde d'un dictionnaire (réalisée en principe par la commande "Ø SAVE". Voir chapitre 10). L'action réalisée par cette option "2" est alors de charger le système d'exploitation, puis le dictionnaire en question, et exécute le mot positionné par AUTORUN lors de la sauvegarde (éventuellement GO si AUTORUN n'a pas été utilisé).
Cette option "2" est donc équivalente à l'option "1", suivie de "Ø RUN".

Avec un MO5 :

Le chargement ne se fait pas automatiquement. Il vous faut donc démarrer en FORTH, puis taper l'ordre "DOS" (D.O.S. signifie "Disk Operating System", traduction anglaise de "Système d'exploitation des disquettes") pour lui faire lire la disquette. Il faut bien entendu qu'au moment où vous tapez cette commande le lecteur soit allumé et contienne une disquette-système.

Après cette opération votre FORTH comporte de nouvelles commandes (décrites ci-dessous), et est modifié en ce sens que le stockage de masse par défaut est la disquette (les mots décrits au chapitre 10 agiront sur la disquette au lieu de la cassette, tant que vous ne le demanderez pas explicitement).
La plupart des mots du chapitre 10 font donc en fait partie du système d'exploitation (FORTH est à ce point de vue plus un environnement de programmation qu'un langage), et il faut considérer les mots de cette annexe comme étant un simple complément à ceux du chapitre 10.

Le système d'exploitation FORTH, BASIC et les autres

En principe, FORTH est un langage conçu pour l'utilisation de disquettes, et la gestion du stockage de masse se fait par blocks "physiques", repérés directement dans le système par leur position sur le disque (numéros de piste et de secteur). Votre ordinateur ne fonctionne pas de cette manière. Il utilise le même principe de stockage que BASIC, et possède un directory (catalogue).

Cela signifie en particulier que chaque block est un fichier de 1 k-octet, composé d'une suite de codes ASCII (et qu'il est lisible en BASIC, par exemple en le considérant comme fichier en accès direct de 25 enregistrements définis (par FIELD) comme des chaines de 40 caractères).
Pour accéder à un block, le système utilise un nom (nom de fichier). Il cherche ce nom dans le directory de la disquette, et y trouve l'emplacement physique du disque où sont écrites les données. En fait, le directory fonctionne exactement comme un carnet d'adresses...

Les inconvénients de la méthode sont un (léger) ralentissement des opérations disque (dû à la recherche dans le directory), mais surtout une capacité du disque limitée à 56 blocks (capacité du directory de la disquette en simple densité). En revanche, cette méthode offre des avantages certains : vous pouvez ranger sur une même disquette des programmes BASIC, des dictionnaires FORTH, des fichiers BASIC ou FORTH (lisibles chacun à partir des deux langages !!!), ou aussi Assembleur, etc.
Notons que seuls les blocks FORTH sont limités à 1 k... Les dictionnaires FORTH et les autres fichiers n'ont pas ce standard.
Une gestion par blocks physiques permettrait de disposer de 80 (ou 160 en double densité) blocks, numérotés de 1 à 80. Pour conserver cette analogie, les blocks utilisables sont aussi repérés par des numéros. Vous pouvez choisir librement ces numéros, les blocks seront sauvegardés avec un nom du type "BLOCKxxx.DAT" où xxx est le numéro choisi. Cela signifie qu'il est recommandé de ne pas utiliser de nombres de plus de 3 chiffres. Si vous tapez le numéro 10000, le fichier aura pour nom "BLOCK100.00T". Et vous n'êtes pas obligés de choisir des numéros consécutifs...

Organisation physique d'une disquette

Les informations contenues sur la disquette sont divisées en 40 pistes (zones circulaires) comportant chacune 16 secteurs. Un secteur peut contenir 128 octets (simple densité) ou 256 octets (double densité). Chaque piste peut donc contenir deux k-octets (quatre en double densité) d'informations. Les pistes sont numérotées de 0 à 39 et les secteurs d'une piste de 1 à 16.

Les informations contenues sur une disquette sont de deux types :
— Les informations que vous désirez écrire sur la disquette (blocks du FORTH, sauvegardes de dictionnaires, programmes BASIC, etc.), qui constituent les informations "utilisateur".
— Les informations système : le système d'exploitation lui-même (et les mots FORTH permettant d'agir sur les disquettes), ainsi que les informations dont il a besoin pour retrouver les informations "utilisateur".

Les informations "utilisateur" sont regroupées en "fichiers" pouvant occuper un certain nombre de secteurs (le dernier n'est pas nécessairement plein, mais un secteur ne peut pas contenir des données appartenant à deux fichiers différents. S'il contient ne fût-ce qu'un octet d'un fichier, il est réservé tout entier à ce fichier).
Le système regroupe ces secteurs en blocs d'un k-octet chacun. En simple densité, un bloc comporte donc 8 secteurs (4 en double densité). Ces blocs (qu'il ne faut pas confondre avec les blocks du FORTH, unités de stockage de masse) sont numérotés de 1 à 8Ø en simple densité (la piste Ø contient les blocs 1 et 2, la piste 1 les blocs 3 et 4, et ainsi de suite), et de 1 à 16Ø en double densité.

Les informations du système occupent trois positions physiques sur la disquette :
Les pistes 0 et 1 contiennent le système d'exploitation et les mots FORTH qui gèrent les disquettes.
La piste 2Ø est la piste "catalogue". C'est là que se trouvent les noms des fichiers, ainsi que leurs caractéristiques et les informations qui permettent de les retrouver sur la disquette.
Les noms occupent les secteurs 3 à 16, et chaque fichier requiert 32 octets (11 pour le nom et l'extension, 1 pour le type de fichier, 1 pour le type de données, 1 pour mémoriser la position du début du fichier sur disquette, et 2 pour le nombre d'octets occupés dans le dernier secteur. Les 16 autres octets ne sont pas gérés par FORTH).
Le secteur 2 de la piste 2Ø contient la table d'occupation de la disquette (FAT, pour "File Allocation Table"). Chacun des octets de cette zone renseigne sur l'occupation d'un bloc de 1 k de la disquette (8 secteurs en simple densité, 4 en double densité). Les blocs consécutifs d'un même fichier y sont chaînés entre eux. Voir le mot FAT ci-après pour des explications complémentaires.
Pour lire un fichier, le système recherche son nom dans la zone

"noms", et lit à partir de la position du début de fichier (lue avec le nom). Si le fichier occupe plus d'un k-octet, la table d'occupation renseigne sur l'endroit où se trouve la suite. Pour le dernier secteur, la zone "noms" (octets 15 et 16) renseigne sur le nombre d'octets à lire. Lors de l'écriture d'un nouveau fichier, le système recherche le prochain bloc libre (dans la table d'occupation). Il ajoute le nom au catalogue, puis écrit le fichier (au fur et à mesure, il marque les blocs écrits dans la table d'occupation). En fin d'écriture, il met à jour les octets 15 et 16 de la zone concernant le nom du fichier dans le directory de la disquette. Cette dernière action est réalisée par le mot IO, et est fondamentale pour la relecture du fichier... N'oubliez donc pas de terminer vos opérations d'écriture par une commande de "fin d'opération" (Voir IO, au chapitre 10).
Le processus est analogue pour l'écriture d'un fichier existant (à cela près que le nom est recherché dans le catalogue...).

(BUFFER) n --- adr

Mot système appelé par le mot BUFFER en version disquette. Son action est d'assigner un buffer au block n. L'adresse adr est celle du premier octet du buffer alloué. Le block n'est pas lu depuis la disquette, mais si le contenu antérieur du buffer était marqué comme modifié, (BUFFER) le réécrit sur disque avant l'allocation.

(DEMIT) c ---

Mot système permettant l'écriture caractère par caractère sur disquette. Avant l'écriture du premier caractère d'un ensemble de données, il faudra initialiser l'opération d'écriture par :

2 (DIO) ou 3 (DIO)

(2 pour un nouveau fichier, 3 pour un fichier déjà existant).
Après avoir écrit le dernier caractère, il faudra terminer l'opération par: 0 (DIO)
La variable D-ERR permet de savoir si l'opération s'est bien déroulée.
(DEMIT) est appelé par le mot PUTC (voir ce mot) via le tableau SYSVEC (position 17).

(DIO) n ---

Mot système qui active le lecteur de disquettes en cours pour une opération d'entrée-sortie.
Avant d'utiliser ce mot, il faut avoir positionné les variables DNAM (adresse du nom du fichier à utiliser), FAT (adresse du buffer de la File Allocation Table), et DMA (adresse du buffer des entrées-sorties disque).
Lors de l'appel de (DIO), le nombre n indique le type d'opération à réaliser.

n = Ø Termine l'opération en cours. Dans le cas d'une écriture, sauvegarde les données restant dans le buffer d'entrées-sorties. Pour une lecture, désalloue les pointeurs vers les données en cours.

n = 1 Initialise une opération de lecture. Recherche le fichier dans le catalogue de la disquette, et lit le premier secteur dans le tampon.

n = 2 Initialise une opération d'écriture dans un nouveau fichier. Crée sur la disquette une entrée au directory.

n = 3 Initialise une opération de réécriture (recherche le fichier dans le catalogue de la disquette).

A la fin des opérations, la variable D-ERR permet de savoir si tout s'est bien passé.
(DIO) est appelé par IO via le tableau SYSVEC (position 15).

(DKEY) --- c

Mot système réalisant la lecture d'un caractère sur la disquette. Le code ASCII du caractère lu est déposé sur la pile. Ce caractère existe toujours, même après une erreur de lecture. Il appartient au programmeur de vérifier la validité de la lecture, en testant la variable D-ERR.
Avant de lire le premier caractère d'un ensemble de données, il faudra initialiser l'opération de lecture par: 1 (DIO).
En pratique, l'appel de (DIO) se fait indirectement, par l'intermédiaire de IO. L'avantage est que IO tiendra compte de la configuration en cours d'utilisation (cassette ou disquette, entre autres).
Pour arrêter la lecture avant la fin des données, il faudra exécuter: Ø (DIO).
(DKEY) est appelé par le mot GETC (voir ce mot) via le tableau SYSVEC (position 16).

(FLUSH) ---

Mot système appelé par SAVE-BUFFERS. Il réalise exactement l'action décrite dans SAVE-BUFFERS. Sa seule raison d'être est la possibilité de vectorisation des entrées-sorties. L'adresse de (FLUSH) est, à la mise sous tension, en position 14 du tableau SYSVEC... Ce qui permet de modifier SAVE-BUFFERS.

B/S --- n

Constante donnant le nombre d'octets par secteur (abréviation de "bytes/sector").
Cette constante vaut 128 pour une disquette en simple densité, et 256 pour une disquette en double densité.

BACKUP n1 n2 ---

Réalise la copie physique de la disquette contenue dans l'unité de disque n1 sur celle contenue dans l'unité n2.
BACKUP ne fait pas le formatage de la disquette contenue dans l'unité n2. Il est nécessaire que la disquette ait été initialisée au préalable (voir FORMAT).
Si vous avez un seul lecteur, tapez "Ø Ø BACKUP". L'ordinateur vous demandera d'introduire les disquettes adéquates au fur et à mesure de la copie.
Attention : BACKUP emploie toute la place disponible en mémoire. Il détruit donc le contenu des buffers. N'oubliez pas de les sauvegarder auparavant (par SAVE-BUFFERS) s'ils contiennent des informations auxquelles vous tenez.

```
Ø 1 BACKUP
```

(Recopie la disquette de l'unité Ø sur celle de l'unité 1).

CASS ---

Positionne la cassette comme stockage de masse.
CASS agit essentiellement au niveau de la vectorisation des entrées-sorties. Il positionne BUFFER comme étant (KBUF), SAVE-BUFFERS comme étant (KFLUSH), IO comme étant (KIO), GETC comme étant (KKEY), et PUTC comme étant (KEMIT).
En outre, il repositionne R/W comme étant (FR/W), pour le cas où vous auriez modifié cette affectation. Cela implique que dans ce cas, il vous faudra refaire votre modification après utilisation du périphérique.

COPY-BLOCKS n1 n2 n3 ---

Recopie les blocks n1 à n2 d'une disquette, en les renumérotant à partir de n3 (jusqu'à n3 + n2 – n1). Lors de l'exécution, COPY-BLOCKS demande quelles sont l'unité source et l'unité de destination. Il est donc possible de copier des blocks d'une disquette sur une autre, ou de changer les numéros des blocks d'une même disquette, en ne changeant pas la disquette lorsque le système le demande... Pour cette dernière opération, il est toutefois plus facile d'employer la commande RENUM.
La copie se fait par l'intermédiaire des buffers de blocks. Elle est donc d'autant plus rapide que le nombre de buffers choisis est grand.

```
10  15  33  COPY-BLOCKS
```

Si on tape 1 comme unité source et Ø comme unité destination, cette commande recopie les blocks 1Ø, 11, 12, 13, 14 et 15 du lecteur 1 sur le lecteur Ø, en les renumérotant respectivement 33, 34, 35, 36, 37, et 38.
Pour recopier sans modifier les numéros, utiliser par exemple :

```
10  15  10  COPY-BLOCKS
```

DENSITY n1 n2 ---

Fixe la densité n1 pour le lecteur n2.

n1 vaut Ø pour signifier la "simple densité", et 1 pour la double densité.

n2 est le numéro du lecteur (de 0 à 3).

Avec un contrôleur de disquettes du type "simple densité", le système fixe automatiquement la densité à Ø (simple). Et il est impossible de lire une disquette écrite en double densité, c'est une question de matériel...
Par contre, un contrôleur du type "double densité" peut lire aussi bien des disquettes en simple densité qu'en double densité. Et si vous avez plusieurs lecteurs de disquettes, FORTH vous donne la possibilité de déclarer que le lecteur 1 doit travailler en double densité, alors que les lecteurs Ø et 2 (si par exemple vous en avez trois) doivent travailler en simple densité...
A vous bien sûr de ne pas vous tromper dans vos manipulations de disquettes, et de ne pas lire par exemple en simple densité une disquette qui a été écrite en double densité...

A l'initialisation, FORTH sélectionne par défaut la "meilleure" densité possible compte tenu du contrôleur (qu'il est capable de reconnaître), pour tous les lecteurs présents. Avec un contrôleur "simple densité", tous les lecteurs sont en simple densité, et vous ne pouvez les changer (si vous avez un tel contrôleur, vous pouvez ignorer purement et simplement le mot DENSITY, il ne peut vous être utile). Avec un contrôleur "double densité", le mot densité peut être utilisé pour lire des disquettes écrites en simple densité.

DISK ---

Positionne la disquette comme stockage de masse.
DISK agit essentiellement au niveau de la vectorisation des entrées-sorties. Il positionne BUFFER comme étant (BUFFER), SAVE-BUFFERS comme étant (FLUSH), IO comme étant (DIO), GETC comme étant (DKEY), et PUTC comme étant (DEMIT).
En outre, il repositionne R/W comme étant (FR/W), pour le cas où vous auriez modifié cette affectation. Cela implique que dans ce cas, il vous faudrait refaire votre modification après utilisation du périphérique.
Si par ailleurs vous disposez de deux lecteurs de disquettes, DISK repositionnera le drive Ø comme drive actif.

DIR n ---

Affiche à l'écran la liste des fichiers de la disquette située dans le lecteur numéro n.
Le format d'affichage d'un fichier est le suivant :

NOM EXTENSION TYPE DE FICHIER TYPE DE DONNÉE PLACE OCCUPÉE

Le *nom* comporte 1 à 8 caractères, et *l'extension* Ø à 3 caractères.
Le *type de fichier* est indiqué par un caractère :

A ..... pour un programme en assembleur.
B ..... pour un programme BASIC.
D ..... pour un fichier de données.
M ..... pour un programme en langage machine.

Le *type de données* est codé par un caractère :

A ..... pour des données ASCII.
B ..... pour des données binaires.

La *taille du fichier* est exprimée en k-octets (chaque fichier occupe au moins 1 k sur la disquette, même s'il ne comporte que trois caractères).

L'écran numéro 73 de l'éditeur apparaît par exemple comme suit:

Nom : BLOCK73
Extension : DAT
Type de fichier : D (données)
Taille : 1
Type de données : A (ASCII)

Une sauvegarde-dictionnaire de numéro 13 apparaît comme suit :

Nom : DICT13
Extension: FTH
Type de fichier: M (langage machine)
Taille: 3
Type de données: B (binaire)

La taille dépend bien sûr de l'importance de l'application sauvegardée...

DRIVE n ---

Sélectionne le lecteur numéro n comme étant l'unité active. Les mots d'accès aux disquettes utiliseront désormais ce lecteur.

```
1 DRIVE 3 EDIT
```

Entre en édition sur l'écran 3 du lecteur 1.

FAT --- adr

Variable système contenant l'adresse d'une zone mémoire où FORTH mémorise les informations sur l'occupation de la disquette. Cette zone comporte 128 ou 256 octets suivant la densité utilisée (simple ou double).

```
FAT @ HEX 80 DUMP
```

Affiche le contenu de la zone (simple densité).
Chaque octet à partir du deuxième, renseigne sur l'occupation d'un bloc de 1 k (8 secteurs en simple densité) du disque. Ne pas confondre ces blocs avec les blocks du FORTH, unités logiques du stockage de masse. La disquette contenant 80 k, seuls les 80 octets à partir du deuxième sont significatifs.
Les octets de la zone pointée par FAT s'interprètent de la façon suivante :

x = 255	Bloc libre.
x = 254	Bloc réservé au système.
192 < x < 201	Bloc de fin de fichier (x – 192 secteurs occupés).
0 < x < 81	Bloc d'un fichier. x est le numéro du prochain bloc du fichier.

Les disquettes sont organisées en 40 pistes de 16 secteurs de 128 octets (256 octets en double densité). En simple densité, chaque piste contient deux blocs.
Le dump ci-dessus donne les résultats suivants (sauf reconfiguration) :
Adresse de la zone FAT : 69BA (en hexadécimal).
Les octets à FF indiquent les blocs libres.
Les octets à FE en 69BA-69BB correspondent au système d'exploitation (piste 0).
Les octets en 69E3-69E4 correspondent à la piste "catalogue" (piste 20).
Les octets à C8 indiquent un bloc saturé (souvent un block FORTH)

FORMAT n ---

Initialise la disquette située dans le lecteur numéro n. Les informations que pourrait contenir cette disquette sont définitivement perdues. *Attention*, ce mot est dangereux !!!. En effet, si vous oubliez le paramètre, FORMAT prendra le nombre qui est sur la pile et fera l'initialisation. En particulier, si votre pile est vide, le nombre sous la pile est en principe 0 (au démarrage du système). FORMAT initialisera

donc la disquette du lecteur Ø, avant d'envoyer un message d'erreur "Pile Vide"...
Si vous êtes d'un naturel distrait, utilisez le mot suivant :

```
: DSKINI    ( --- )
        1Ø  Ø  DO  BELL  4Ø  WAIT  RED  FRAME  BELL
                   4Ø  WAIT  CYAN  FRAME
               LOOP
        CR  CR  ." Numéro du lecteur pour initialisation : "*
        INPUT  DROP  FORMAT  CR  CR  ;
```

KILL n ---

Supprime le block logique (block FORTH) numéro n de la disquette située dans l'unité active. En fait, seule la piste "catalogue" de la disquette est mise à jour.

```
1 DRIVE 13 KILL
```

Supprime le block 13 de la disquette du lecteur 1.

RENUM n1 n2 n3 ---

Renumérote les blocks consécutifs de n1 à n2 de la disquette active à partir du numéro n3. A la différence de COPY-BLOCKS, les blocks ne sont pas réécrits (seuls les numéros sont modifiés dans le catalogue). En revanche, RENUM ne peut agir que sur une disquette.
Une pratique raisonnable est de réserver les blocks 1 à 9 pour les "brouillons" de vos applications (zone de travail usuelle), puis, lorsque vous estimez être arrivés à un degré satisfaisant de finition, renuméroter ces blocks. Vous pouvez même prendre aussi l'habitude de réserver les blocks 1Ø, 2Ø, 3Ø, etc. pour des écrans de chargement (voir LOAD, chapitre 10).

```
1 7 21 RENUM
```

Renumérote les blocks 1 à 7 avec les numéros 21 à 27.

SEC-RD n1 n2 ---

Réalise la lecture du secteur n2 de la piste n1 de la disquette.
Le résultat de la lecture se trouve à l'adresse donnée par la variable DMA.

```
2Ø 3 SEC-RD DMA @ B/S DUMP
```

"dumpe" le premier secteur du directory de la disquette du drive courant...
Vous devriez voir apparaître les noms des 4 premiers fichiers du catalogue.
Il est intéressant d'étudier comment sont codées les caractéristiques des fichiers.
(Voyez le mot DIR, et faites des essais en utilisant les mots FASC! et FTYP!...)

SEC-WR n1 n2 ---

Ecrit en piste n1, secteur n2 le contenu de la zone DMA (128 ou 256 octets à partir de l'adresse donnée par la variable DMA, suivant la densité positionnée pour le drive en cours).

```
12 2 DDUP Ø DRIVE SEC-RD 1 DRIVE SEC-WR
```

Recopie le secteur 2 de la piste 12 de la disquette dans le lecteur Ø sur le secteur correspondant de la disquette du lecteur 1.
Si vous utilisez cette méthode pour jongler avec les disquettes, n'oubliez pas de mettre à jour la FAT...

Annexe C

Structure du dictionnaire

C.1 Généralités sur l'interpréteur Description d'un mot

L'interpréteur entre en action lorsque vous avez tapé la touche ENTREE, ou lorsqu'il y est invité par une commande comme LOAD. Son premier travail consiste à isoler les mots du flot d'entrée (ce que vous avez tapé au clavier, ou le contenu d'un block). Puis, pour chacun des mots trouvés, il exécute l'action requise.
Il a donc besoin, pour travailler, de deux types de renseignements :
— Des renseignements permettant la recherche des mots par leur nom.
— Un mot étant trouvé, une description des actions à exécuter.
FORTH est original à ces deux points de vue.

En ce qui concerne la recherche :
Les mots forment ce qu'on appelle en informatique une liste chaînée (en fait, plusieurs listes chaînées, une par vocabulaire. Voir ci-dessous). Cela signifie que chaque mot comporte dans sa définition (dans le dictionnaire) une indication sur l'endroit où trouver le mot suivant (son adresse).

C'est dire que la définition de chacun des mots du dictionnaire comporte:
Une *zone* pour mémoriser son nom. Cette zone s'appelle le "champ nom", ou en anglais son "Name Field" (en abrégé nf). Cette zone est indispensable pour que l'interpréteur puisse comparer le nom à celui du flot d'entrée...
Un lien de chaînage vers le mot suivant à consulter, si le mot actuel n'est pas le bon.
La zone correspondante, le "champ lien" s'appelle en anglais le "link field", et s'abrège habituellement en lf. Le lf d'un mot contient l'adresse du nf du mot suivant à consulter si la recherche n'est pas terminée.
On abrège souvent
"adresse du nf" en *nfa* (name field address)
"adresse du lf" en *lfa* (link field address).
Un bon exemple de parcours du dictionnaire est fourni par le mot VLIST. Le parcours se fait à partir de la fin du dictionnaire, en allant "à reculons", ce qui donne l'impression que les différentes définitions sont empilées l'une au-dessus de l'autre.
VLIST consulte la variable CONTEXT pour obtenir (indirectement) le nfa du dernier mot défini dans le vocabulaire sélectionné. Il se rend à cette adresse, affiche le nf, et consulte le lf pour avoir le nfa du mot suivant à afficher. Et ainsi de suite... Le lf du dernier mot du FORTH, LIT (le premier du dictionnaire, puisqu'on va en marche arrière), contient le nombre Ø, qui n'est pas l'adresse d'un mot du dictionnaire et marque la fin du parcours.

En ce qui concerne l'exécution :
Un ordinateur n'est jamais capable d'exécuter que du "code machine", c'est à dire une suite d'octets contenant des instructions pour son microprocesseur.
La définition de chaque mot doit donc comporter aussi une zone "code machine à exécuter". En fait, elle contient non pas le code à exécuter, mais l'adresse où se trouve ce code. La zone du nom contenant cette adresse s'appelle en anglais "code field" et s'abrège en *cf.* L'adresse du cf (adresse de l'adresse du code à exécuter) est le *cfa* (code field address).
Et lorsque FORTH doit exécuter un mot (en mode interactif), il place son cfa sur la pile et appelle le mot EXECUTE, dont l'action est justement d'exécuter le mot dont le cfa est sur la pile.

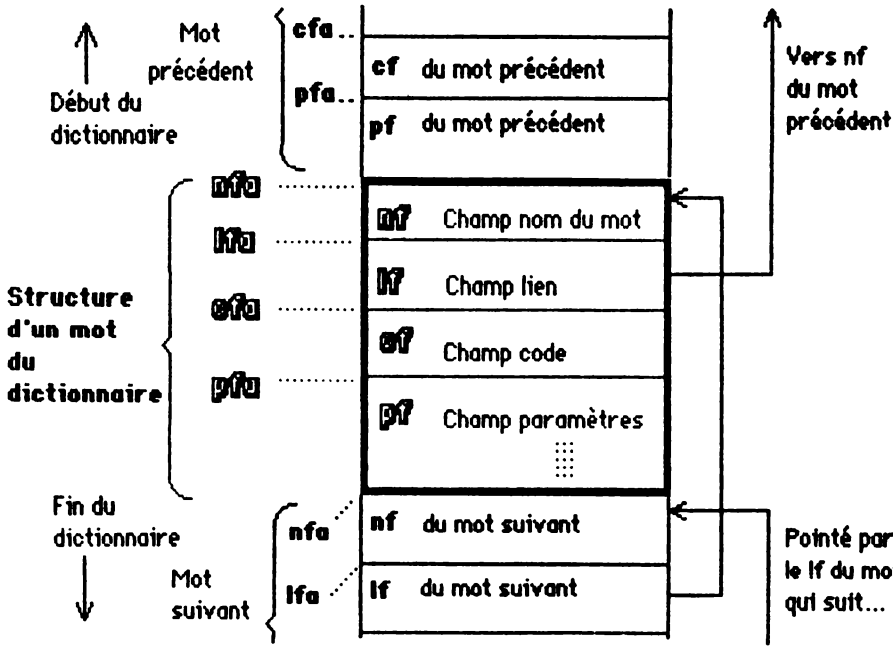
Mais il arrive souvent qu'un mot X en appelle un ou plusieurs autres (cas des mots définis par :..;)... C'est simplement alors que l'action du code machine pointé par son cf est d'exécuter successivement les mots qui le constituent. Il est donc nécessaire de mémoriser aussi les cfa des mots qui ont servi à définir le mot X. Ces cfa sont des paramètres pour la routine d'exécution du mot X. La zone de la définition de X qui les contient s'appelle la "zone paramètres", en anglais "parameter field" (abrégé en *pf*). L'adresse du pf de X se note le *pfa* de X (parameter field address).
C'est une des plus grandes originalités de FORTH... Les mots sont en fait regroupés par type de définition. Tous les mots définis par :..; ont le même cf, et ne diffèrent que par leurs pf respectifs. Mais le pf d'un mot peut contenir autre chose qu'une suite de cfa, tout dépend de la routine d'exécution du mot. Par exemple, une constante aura pour pf sa valeur, et pour cf l'adresse d'une routine dont l'objet est de déposer sur la pile le contenu du pf. Pour un mot "en langage machine" , le code machine à exécuter est dans son pf, et le cf du mot contient donc son propre pfa.
En fait, seuls les mots fondamentaux sont écrits en langage machine, la plus grande partie du FORTH est écrite... en FORTH !!!

En résumé, chaque définition du dictionnaire comporte 4 zones :

n f Nom du mot
l f Lien vers le mot précédent du dictionnaire (nfa du mot précédent).
c f Code machine à exécuter lors de l'appel du mot.
p f Paramètres utilisés par le code machine.

Les deux premières zones, utilisées par l'interpréteur lors de la recherche, et la troisième, qui renseigne sur le type du mot, constituent la partie en-tête de la définition.
Dans votre FORTH, ces zones sont contigües, ce qui n'est pas obligatoire.

En laissant de coté la question des vocabulaires (voir chapitre 4), l'organisation du dictionnaire en mémoire est la suivante (adresses croissantes vers le bas) :



C.2 Les différentes zones d'une définition

Champ nom

Le nf d'un mot comporte, outre le nom du mot, quelques renseignements utilisés lors de la compilation (lorsqu'une recherche est de toute façon nécessaire). Le nom du mot est codé en ASCII, avec le bit 7 à Ø, sauf le dernier caractère, dont le bit 7 est à 1. C'est pour cette raison qu'un nom de mot ne peut contenir de caractère accentué (les caractères accentués sont codés par des codes ASCII supérieurs à 128, et ont donc leur bit 7 à 1). Ce nom est précédé d'un octet "de longueur". Mais puisque la longueur maximum d'un nom est 31 carac-

tères, seuls les 5 bits de poids faibles sont utilisés pour coder la longueur. Les autres ont l'utilité suivante :
le sixième bit (le bit 5) s'appelle le "bit de smudge". Il sert essentiellement à interdire la récursivité (FORTH, n'ayant que des variables globales, n'accepte pas la récursivité au sens le plus général). Il est mis à 1 au début de la compilation d'une définition, et remis à zéro à la fin. Lorsque le bit de smudge est à 1, le mot ne peut être trouvé dans le dictionnaire...
Le septième bit (le bit 6) est ce qu'on pourrait traduire par "bit de priorité" (en anglais "precedence bit"). Ce bit est à 1 lorsque le mot est immédiat, c'est-à-dire lorsqu'il doit s'exécuter pendant la phase de compilation.
Le huitième bit est toujours à 1, pour marquer le début de la définition. Il permet au système de retrouver le début du nf en ne connaissant que le lf du mot (voir TRAVERSE).
Par exemple, le nf de VLIST est composé des octets suivants (en hexadécimal) :

85 56 4C 49 53 D4

85 indique un mot non immédiat, et un nom de 5 caractères. 56, 4C, 49 et 53 sont les codes ASCII respectifs de V, L, I, et S. D4 est le code ASCII de T, avec le bit 6 positionné à 1 (le code ASCII de T est 54).

Le nf de BEGIN est composé de :

C5 42 45 47 49 CE

C5 indique que BEGIN est un mot immédiat...

Pour un nom de n caractères, le champ nom occupe donc n + 1 caractères au dictionnaire.

Champ lien

Le champ lien est une adresse, donc occupe invariablement 2 octets. Le lf du premier mot du dictionnaire (le dernier lu dans le cas d'une recherche infructueuse) contient Ø. Pour avoir une idée de la rapidité du processus, dites-vous que lorsque vous tapez un mot qui ne figure pas au dictionnaire, entre le moment où vous tapez ENTREE et l'affichage du message d'erreur, l'interpréteur a eu le temps de parcourir tout le dictionnaire, comparant votre nom aux quelques 5ØØ mots du FORTH... Et encore, cela fait, il vérifie que votre mot n'est pas un nombre...

Champ code
Occupe 2 octets (c'est une adresse). Contient l'adresse d'exécution du mot de définition qui a servi à définir le mot considéré. Par exemple, tous les mots définis par ":" ont le même cf (11AF pour un TO7 ou un TO7-70). Les autres mots de définition classiques sont VARIABLE, CONSTANT, DVARIABLE, DCONSTANT, TABLE, ARRAY, CTABLE, CARRAY, STRING, ARRAY$, MUSTAB, CHARTAB...
Et surtout, grâce aux mots CREATE et DOES > , vous pouvez définir vos propres mots de définition...

Champ paramètres
Peut contenir des objets divers, en fonction de la routine d'exécution. Pour un mot défini par ":", il s'agit de la suite des cfa des mots qui constituent la définition. Dans ce cas, les nombres et les chaînes de caractères compilés comme littéraux sont précédés de cfa spécifiques (LIT ou (")), pour qu'il ne soient pas interprétes comme des cfa eux-mêmes.
Pour un mot écrit en code machine, le pf contient en général le code machine lui-même (et est pointé par le cf).
Pour la plupart des autres mots, qui sont des mots de structuration des données, le pf contient les données elles-mêmes.
Par exemple pour une chaine de caractères définie par le mot STRING, le pf contient successivement :
— un octet pour la longueur maximum.
— un octet pour la longueur actuelle.
— les codes ASCII des caractères de la chaine.
Si la chaine n'a pas la longueur maximum, les derniers caractères sont indéterminés. La routine d'exécution d'un tel mot consiste à déposer sur la pile l'adresse du début de la chaine (pfa + 2) et l'octet de longueur (octet en pfa + 1)...

C.3 L'interpréteur externe et l'interpréteur interne
La pile des retours

Le schéma de fonctionnement décrit ci-dessus est celui de l'interpréteur externe, celui qui cherche les mots au dictionnaire, et suivant le mode (voir STATE) les compile ou les exécute. On l'appelle pour cette raison "l'interpréteur de mots".
Notons au passage que si le mot n'est pas trouvé au dictionnaire, l'interpréteur essaie de le considérer comme un nombre (compte tenu de la base en cours). Ce n'est que s'il n'y parvient pas qu'il envoie un message d'erreur. S'il y parvient, il compile ce nombre comme littéral ou le dépose sur la pile (suivant le mode indiqué par STATE).

Lors de l'exécution, pour traiter les mots l'un après l'autre, il est nécessaire que le système mémorise l'adresse du prochain mot à exécuter, au retour de l'exécution du mot en cours. Il utilise pour cela un pointeur de programme, et la pile des retours.
On a par exemple défini :

```
: TOUR RED FRAME ;
: ALERTE TOUR BELL ;
```

Et on demande l'exécution du mot ALERTE. Le cfa de ALERTE demande d'exécuter TOUR, puis BELL (dont les cfa composent le pf de ALERTE). La première action à exécuter est donc TOUR, qui est défini de la même manière (RED, puis FRAME).
Le premier cfa lu dans le pf de TOUR est celui de RED. Mais avant d'exécuter RED, FORTH dépose sur la pile des retours l'adresse du prochain mot à exécuter dans ALERTE (c'est bien là qu'il devra aller au retour de TOUR) et met dans le pointeur de programme l'adresse où se trouve le cfa de RED, dans le pf de TOUR. Il peut alors aller exécuter le mot RED. Cela fait, il incrémente de 2 le pointeur de programme, et remplace donc l'adresse de RED par celle de FRAME. Il exécute alors FRAME après avoir sauvegardé l'adresse de ;S (mot compilé au dictionnaire par ";"). Lorsqu'après avoir exécuté FRAME il en arrivera à ;S, l'action qu'il aura à exécuter sera de dépiler l'adresse du mot qui a appelé TOUR, et de la mettre dans le pointeur de programme, et ainsi de suite... A chaque fois qu'il aborde un mot défini par ":", il mémorise sur la pile des retours l'adresse du mot appelant avant de mettre à jour le pointeur de programme.

Cet ensemble de manipulations est réalisé par ce qu'on appelle l'"interpréteur interne", et qui ne correspond pas à un mot en particulier. Une partie du travail est accomplie par le mot ";S", et une autre par la routine d'exécution de ":"...

Soyez donc prudent lorsque vous jouez avec la pile des retours. Les valeurs que vous pouvez y déposer par > R s'y trouvent mélangées avec les adresses de retour, au fur et à mesure des opérations. Comme chaque empilage correspond à un dépilage, le système s'y retrouve, sauf si vous avez oublié de dépiler votre nombre par R > avant la fin de la définition. Dans ce cas, à la fin de l'exécution de votre mot, l'interpréteur interne dépilera votre nombre et l'interprètera comme une adresse de retour. Et la probabilité pour que ce qu'il y trouvera corresponde à quelque chose de sensé est extrêmement faible...

Autre conséquence de ce qui précède : si vous écrivez un mot en langage machine, le code doit se terminer par un branchement sur l'interpréteur. Le pointeur de programme est en fait le registre Y du 68Ø9. Une méthode est de terminer votre partie en langage machine (dans le pf de votre mot) par le code suivant : AE A1 6E 94
Ces quatre octets correspondent aux instructions suivantes :

```
LDX   ,Y++    cfa dans X, et mise à jour du pointeur de programme
JMP   [,X]    et on va exécuter la routine pointée par X...
```

C.4 Les vocabulaires

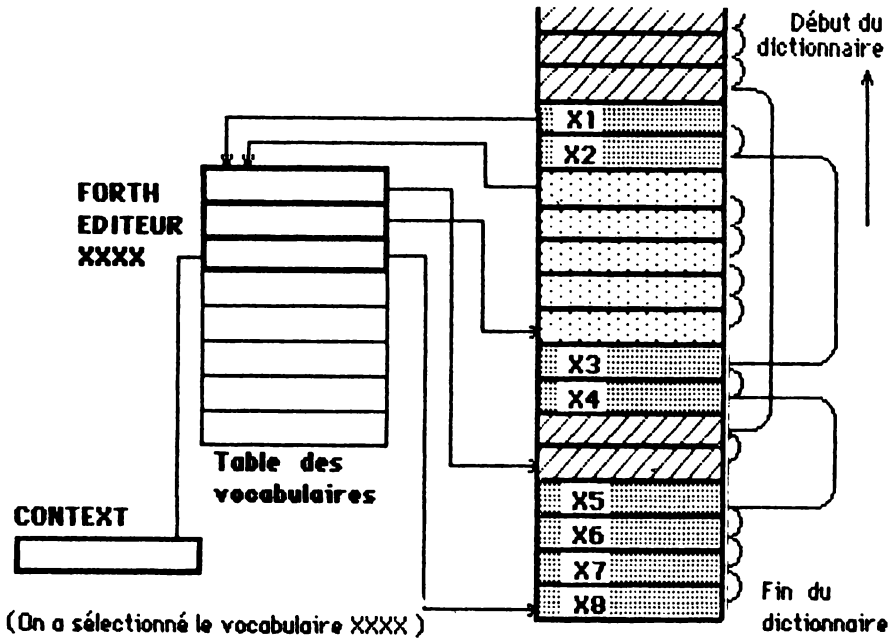
Les mots du dictionnaire présentés dans les chapitres 1 à 11 sont les mots fondamentaux du langage. Les mots de l'éditeur, en particulier, n'y figurent pas. Pour les faire apparaître dans un VLIST, par exemple, il est nécessaire de positionner l'éditeur comme vocabulaire actif pour la recherche-dictionnaire.
Le dictionnaire est organisé en vocabulaires, qui sont en fait des sous-dictionnaires disjoints, considérés chacun comme une extension du vocabulaire de base (FORTH).

Si vous écrivez une application comportant de nombreux mots spécifiques, il est commode d'en faire un vocabulaire, puis d'écrire un mot FORTH qui ouvre ce vocabulaire et appelle le mot final de l'application.
L'interpréteur emploie deux variables, CONTEXT et CURRENT, pour gérer les vocabulaires.
CONTEXT indique le vocabulaire dans lequel doivent se faire les recherches-dictionnaire (contient le nf du dernier mot du vocabulaire), et CURRENT celui dans lequel seront insérées les prochaines définitions.
Les définitions d'un même vocabulaire sont chaînées entre elles de la manière suivante :
le lf d'un mot contient le nfa du mot précédent du même vocabulaire, et le lf du premier mot d'un vocabulaire autre que FORTH permet d'obtenir (voir ci-après) le nfa du dernier mot du FORTH.
C'est ce qui explique que si on tape EDITOR VLIST les mots de l'éditeur apparaissent en premier. Les mots d'un vocabulaire ne sont pas nécessairement contigus dans le dictionnaire, ils viennent s'y placer au fur et à mesure des définitions...
Le fonctionnement interne du système utilise une table en page zéro (en RAM), dans laquelle chaque vocabulaire occupe 4 octets. Vous pouvez employer au maximum 8 vocabulaires en RAM, compte tenu de FORTH et de l'éditeur (table de 32 octets).
Les 4 octets d'un vocabulaire sont constitués de la manière suivante : les deux premiers contiennent les codes hexadécimaux 81 et AØ, constituant un "faux nf" (longueur 1, nom constitué d'un espace). Ce nf étant impossible dans une définition (car l'espace est un séparateur, et ne peut faire partie d'un nom), cela n'agit pas sur le fonctionnement du système. L'utilité de ces deux octets est que les deux octets suivants sont alors considérés par les mots de recherche comme un lf.
Ces deux octets suivants contiennent le nfa du dernier mot défini dans le vocabulaire correspondant, permettant de faire le lien avec les mots du vocabulaire.
Le lf du dernier mot du vocabulaire pointe sur le premier octet de la zone de la table correspondant au vocabulaire FORTH.

Lorsqu'on tape : 2 C/L ! EDITOR VLIST (liste des mots en colonne), on voit d'ailleurs apparaître le "faux mot" constitué d'un espace après le mot LK (qui est le premier mot de l'éditeur). En effet, le lf de LK pointe sur ce faux mot (table des vocabulaires), dont le lf renvoie au dernier mot du FORTH...

La variable CONTEXT contient l'adresse du troisième octet de la zone de la table des vocabulaires relative au vocabulaire sélectionné. Autrement dit, l'expression CONTEXT @ @ donne le nfa du dernier mot défini dans le vocabulaire sélectionné.
Exemple : EDITOR CONTEXT @ @ ID. EDSCR OK

L'exemple ci-dessous montre l'organisation des chaînages :



On a défini un vocabulaire XXXX, dans lequel on a défini successivement les mots X1, X2,..., X8.
La définition d'un nouveau vocabulaire a été faite par VOCABULARY XXXX.
Pour que les nouvelles définitions mettent à jour la table des vocabulaires, on a ensuite tapé XXXX DEFINITIONS (fait pointer la variable CURRENT vers le troisième octet de la zone XXXX de la table des vocabulaires. Le nfa de chaque nouveau mot défini vient s'écrire à l'adresse pointée par CURRENT).

On a ensuite défini X1 et X2, puis tapé EDITOR DEFINITIONS, etc. Le lf de X1, comme celui du premier mot de l'éditeur, pointe sur le début de la zone FORTH de la table.
Pour positionner XXXX comme contexte (et donc pour pouvoir utiliser les mots de XXXX), on a simplement tapé XXXX.

Le mot XXXX est un mot du dictionnaire, dont l'action est de positionner la variable CONTEXT avec l'adresse du troisième octet de la zone XXXX de la table des vocabulaires.
Le pf des mots FORTH, EDITOR, et XXXX contient entre autres (octets 3 et 4) un lien vers le vocabulaire précédent (chaînage tranversal). La variable VOC-LINK contient le point de départ de ce chaînage (pointe vers le dernier vocabulaire créé).

Exemple : affichage de la liste des vocabulaires disponibles...

```
: VOC?  ( --- )
     VOC-LINK
     BEGIN  @  DUP
     WHILE DUP 2-  NFA  5 SPACES ID.   (2- donne le pfa)
     REPEAT DROP ;
VOC?  XXXX  EDITOR  FORTH OK
```

Annexe D

Les mots système
Utilisation de la vectorisation

Il n'est pas question ici de détailler l'ensemble des mots système. Peut-être pensez-vous qu'il y en a beaucoup, mais c'est une tradition du FORTH... De même que c'est un langage "adulte" en ce sens qu'il laisse à l'utilisateur l'entière responsabilité de ses erreurs (et ne fait aucun contrôle), FORTH, qui est un langage à la fois évolué et proche de la machine sur laquelle il est implémenté, ne cache pas ses mots de très bas niveau (mots système).

Les débutants n'utiliseront en général pas les mots système, et les connaisseurs n'auront pas de difficulté pour comprendre l'action réalisée par les mots pour lesquels les explications de ce manuel sont un peu superficielles... Il est si facile de décompiler un mot FORTH, ou d'obtenir son dump hexadécimal si c'est un mot machine (les routines en langage machine sont en général très courtes, et peuvent facilement être désassemblées à la main).

Une des qualités de FORTH est par ailleurs la facilité avec laquelle on peut adapter ou modifier les actions réalisées. Ce point est particulièrement sensible en ce qui concerne les entrées-sorties.
Par exemple, il est commode qu'une application réalisant un affichage puisse écrire de manière plus ou moins banalisée à l'écran, à l'imprimante, ou sur disque. Ce genre de résultat peut s'obtenir par des moyens divers, mais le plus simple est certainement que la routine utilisée pour l'affichage d'un caractère (ou de manière plus banalisée l'envoi d'un caractère à un périphérique) soit **indirecte.** Cela signifie que le mot EMIT (le mot habituel pour l'affichage d'un caractère) réalise en fait l'appel d'un autre mot, et a une action différente suivant le mot appelé. Cela se fait en FORTH en plaçant le cfa du mot à appeler à une adresse fixe, et en remplaçant le nombre à cette adresse avant l'utilisation de EMIT.
Le contenu de cette adresse doit pouvoir être modifié, donc cette adresse doit être en RAM (alors que le mot EMIT lui-même est en ROM).
C'est ce procédé qu'on appelle la **vectorisation** de EMIT.
De la même manière, il est souhaitable de pouvoir modifier l'action de certains mots fondamentaux, en vue de prolongements possibles. Par exemple, si on veut implémenter des nombres flottants, la routine d'analyse des nombres doit être modifiée, ainsi que l'interpréteur.

Les adresses de vectorisation des mots concernés sont regroupées dans le tableau SYSVEC. En réalité, SYSVEC n'est pas exactement un tableau au sens habituel du FORTH (il est en page zéro), mais pour l'utilisateur il se comporte exactement de la même manière.
La structure du tableau est la suivante :

Position	Mot appelant	Mot appelé par défaut	Action
0	CR	(CR)	Passage à la ligne à l'écran (mode CONSOLE)
1	EMIT	(EMIT)	Affichage d'un caractère à l'écran (mode CONSOLE)
2	KEY	(KEY)	Lecture d'un caractère sur un périphérique
3	NUMBER	(NBR)	Conversion d'une chaîne de caractères en nombre double longueur
4	INTERPRET	(INTRP)	Interpréteur
5	ABORT	(ABORT)	Traitement des erreurs
6	GO		...Mot appelé au démarrage du FORTH
7	COLD		...Réinitialisation du système
8	...	(PKEY)	Lecture d'un caractère (mode série RS-232)
9	EMIT	(PEMIT)	Affichage d'un caractère à l'imprimante (mode PRINTER)
10	CR	(PCR)	Passage à la ligne (mode PRINTER)
11	SCRPRT	(GPRT)	Copie d'écran graphique
12	R/W	(FR/W)	Lecture d'un block (cassette ou disque)
13	BUFFER	(BUFFER) ou (KBUF)	Allocation d'un buffer pour un block (suivant version, disque ou cassette)
14	SAVE-BUFFERS	(FLUSH) ou (KFLUSH)	Sauvegarde de(s) block(s) modifié(s) (un seul pour (FLUSH), ou tous pour (KFLUSH))
15	IO	(DIO) ou (KIO)	Initialisation d'un périphérique pour une opération d'entrée-sortie
16	GETC	(DKEY) ou (KKEY)	Lecture d'un caractère sur un périphérique (utilisé par les mots concernant le stockage de masse)
17	PUTC	(DEMIT) ou (KEMIT)	Ecriture d'un caractère sur un périphérique (utilisé par les mots concernant le stockage de masse)
18	ERR	...	Table des messages d'erreur. Voir MESSAGE
19	...	CMD	Table des commandes de l'éditeur

Vous pouvez pour chacune de ces fonctions remplacer dans SYSVEC le cfa du mot par défaut par le cfa du mot de votre choix, pour modifier le comportement du mot appelant.
Bien sûr, c'est à vos risques et périls. Le comportement des mots qui font eux-mêmes appel à ces mots appelants sera également modifié.

Mais si vous remplacez par exemple l'écriture d'un caractère sur disquette (PUTC) par l'écriture d'un caractère à l'écran ou sur disque dur, vous n'aurez pas de problème, le tout est d'être raisonnable.
Par exemple : FIND (EMIT) 17 SYSVEC !
PUTC utilise désormais (EMIT)
La conséquence la plus amusante est que maintenant SAVE-BUFFERS sauvegardera les buffers sur l'écran... !!!
Les 11 premiers chapitres de ce manuel donnent de nombreux exemples d'utilisation de cette vectorisation. Par ailleurs, cette technique est très riche en possibilités, et peut améliorer dans une large mesure la souplesse de vos applications. Elle permet en particulier d'écrire des mots qui feront appel lors de leur exécution à des mots qui ne sont pas encore définis.

Annexe E

Organisation de la mémoire

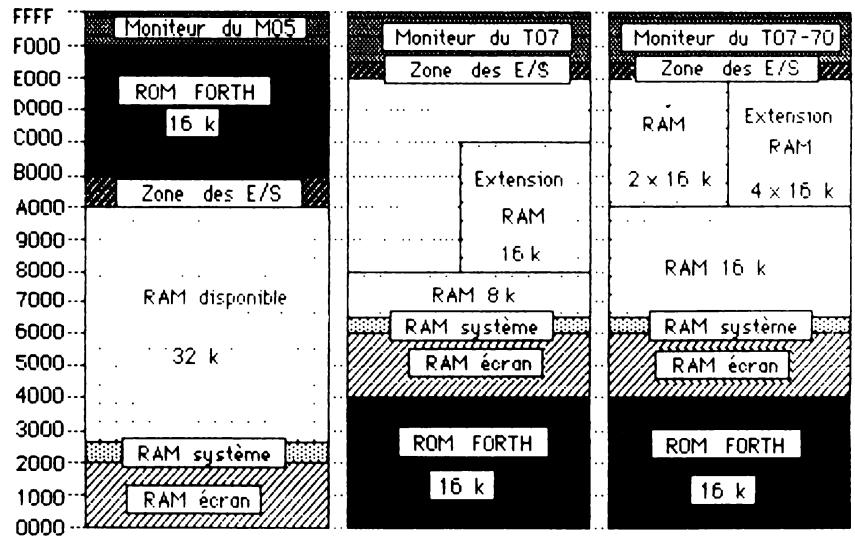
E.1 Généralités

Votre ordinateur configure sa mémoire en fonction de la mémoire disponible, et des périphériques présents. Cette configuration se fait automatiquement dès la mise sous tension dans le cas d'un TO7 ou d'un TO7-70, ou d'un MO5 sans disquettes. Si vous disposez d'un MO5 et d'un (ou plusieurs) lecteur(s) de disquettes, la configuration effective ne se fait que lorsque vous tapez l'ordre DOS.

On peut distinguer en première approche six zones mémoire :
Le moniteur de l'ordinateur. ROM de 4k (MO5) ou 6k TO7 ou TO7-70) contenant les routines de base de l'ordinateur, indépendamment du langage.
Exemples : lecture d'un caractère au clavier, démarrage du système...
La zone des entrées/sorties (4k pour le MO5, 2k pour TO7 ou TO7-70).
C'est à ces adresses que prennent physiquement place les interfaces des divers périphériques que vous pouvez connecter.
Exemples : contrôleur de disquettes, PIA du contrôleur de communication.
La cartouche-ROM FORTH. (16k ROM). Contient la partie ROM du langage.

La mémoire de l'écran (8k d'adresses)
En fait, cette zone contient deux "pages" de 8k de mémoire, aux mêmes adresses.
Une page "forme" mémorisant les points qui sont en couleur "fond" ou "point".
Une page "couleur" mémorisant octet par octet les couleurs de point et de fond.
Chacun de ces octets correspond à un segment horizontal de 8 points, ce qui explique les particularités de la gestion des couleurs.
La RAM système. Voir description plus loin.
La RAM disponible. Voir description plus loin.

La figure ci-dessous décrit l'occupation générale de la mémoire, suivant votre matériel.
Les adresses (à gauche) sont en hexadécimal.



Sur le MO5, le moniteur occupe les adresses F000 à FFFF, et la zone des entrées-sorties les adresses A000 à AFFF. Sur le TO7 et le TO7-70, ces mêmes zones occupent respectivement les adresses E800 à FFFF, et E000 à E7FF.

E.2 La mémoire utilisateur

Elle commence à la fin de la RAM système, et sa taille dépend à la fois du type de matériel et de la présence (ou de l'absence) d'extension mémoire. Les chiffres ci-dessous concernent la taille totale de la RAM ... **(RAM = RAM système + RAM utilisateur)**

MO5.....32k (adresses 2000 à 9FFF). Pas d'extension possible.

TO7.....8k sans extension (adresses 6000 à 8000).

.....24k avec boitier d'extension de 16k (adresses 6000 à BFFF).

TO7-70.....48k sans extension. L'organisation de ces 48k est un peu particulière, elle utilise la technique dite de "bank-switching" :
les 16k aux adresses 6000 à 9FFF sont bien uniques, mais les 16k suivants (adresses A000 à DFFF) sont "doublés", et constituent en fait une zone mémoire de 32 k, 2 fois 16k aux mêmes adresses...
On peut accéder à l'une ou l'autre des "banques" de 16k par le mot BANK (paramètre 0 ou 1). Essayez par exemple :
0 BANK 10 EDIT et tapez un texte quelconque, puis la touche STOP.
1 BANK 10 EDIT et tapez un autre texte, puis la touche STOP.
0 BANK 10 LIST vous donne le premier texte.
1 BANK 10 LIST vous donne le deuxième texte...

.....112 k avec l'extension mémoire de 64k.
Cette extension est structurée comme les 32k aux adresses hautes de la RAM, en 4 banques de 16 k. On dispose donc avec cette extension d'une RAM organisée en 16k de base (adresses 6000 à 9FFF), et 6 banques de 16k , en parallèle aux adresses A000 à DFFF.
Le paramètre du mot BANK a pour valeurs possibles 0, 1, 2, 3, 4 ou 5.

Pour connaître effectivement la taille de la mémoire utilisateur, il suffit de savoir qu'à la mise sous tension, FORTH teste la RAM et place les buffers de blocks aux adresses les plus hautes (dans le cas du TO7-70 dans la banque Ø). La première adresse non disponible est donnée par le mot LIMIT.
Exemple: `LIMIT HEX U. C000 OK`
(TO7 avec extension 16k. La RAM s'étend de 6000 à BFFF).

La RAM utilisateur (au dessus de la RAM système) est organisée comme suit :

	LIMIT
Buffers de blocks (1k par buffer)	
	FIRST
RAM non utilisée	
Zone de stockage des chaînes de caractères temporaires (255 octets).	
PAD	
84 octets réservés aux conversions numérique-alphanumérique	
	HERE (ou DP @)
Dictionnaire en RAM (les mots que vous avez définis)	
RAM système. Voir E-3	

La taille de la RAM effectivement disponible pour les nouvelles définitions au dictionnaire est donnée par: `FIRST PAD @ - 255 - .`
Au fur et à mesure des nouvelles définitions, la valeur donnée par HERE augmente, et la zone de stockage temporaire est décalée d'autant. La mémoire est considérée comme saturée lorsque la fin de cette zone dépasse l'adresse donnée par FIRST. Il vous est dans

ce cas possible de diminuer le nombre de buffers utilisés pour récupérer de la place. Bien entendu, il est indiqué de sauvegarder auparavant les buffers que vous venez d'utiliser, car le mot BUFFERS réinitialise l'ensemble de l'allocation...
MAXBUF donne à tout instant le nombre maximum de buffers qu'il est possible de choisir, c'est-à-dire en gros le nombre de k-octets entre la fin de la zone de stockage des chaînes temporaires et le haut de la mémoire utilisateur (en fait, un buffer occupe 1Ø28 octets...). A la mise sous tension, le nombre de buffers est fixé à 4 pour un système de plus de 16k, 2 dans le cas contraire.

La zone de stockage des chaînes temporaires est utilisée par les mots INPUT, LINPUT, INPUT$, LINPUT$, " et ." (en mode interactif), TEXT, etc. Elle commence à l'adresse donnée par PAD. L'octet à l'adresse PAD est l'octet de longueur de la chaîne.

La zone des conversions numériques se termine un octet en dessous de l'adresse donnée par PAD. C'est là que se font tous les calculs de conversion d'un nombre (sous forme interne) en chaîne de caractères affichable.

E.3 La RAM système

Elle commence à l'adresse 6ØØØ sur TO7 et TO7-70, et à l'adresse 2ØØØ sur MO5.
Sa taille dépend de la présence ou de l'absence de lecteurs de disquettes. Elle comporte (les adresses ci-dessous sont valables pour un TO7 ou un TO7-70 ; pour un MO5, remplacer 6ØØØ par 2ØØØ, etc.) :

La page zéro du moniteur (adresses 6ØØØ à 6ØFF)
C'est ici que sont rangés tous les paramètres utilisés par le moniteur de l'ordinateur (par exemple la position du curseur, la taille de l'affichage, les paramètres musicaux, etc.), qui ne sont pas gérés directement par le langage...

La page zéro du FORTH (adresses 6100 à 61FF)
Cette zone contient les valeurs des paramètres nécessaires au fonctionnement interne du langage, et les valeurs des variables du dictionnaire ROM (qui ne peuvent bien sûr pas être en ROM).

Le buffer des entrées au clavier (adresse 6200, pointée par la variable TIB).
D'une longueur de 84 octets, cette zone est celle où sont rangées les données tapées au clavier. La frappe de la touche ENTREE provoque l'interprétation de son contenu, et sa remise à zéro (dans l'attente d'une nouvelle entrée).

Le nom du fichier à utiliser par défaut.
Zone de 11 octets contenant le nom de fichier qui sera utilisé pour les prochains accès au stockage de masse. Ce nom est automatiquement positionné par les mots du FORTH qui sont standards (BLOCK, SAVE-BUFFERS, etc.), mais vous pouvez le positionner vous-même (voir FNAM!, FNB!) pour utiliser les mots permettant un accès caractère par caractère au stockage de masse (voir IO, GETC, PUTC...). Le nom comporte deux parties : le nom proprement dit occupe les 8 premiers caractères, et une extension, qui occupe les trois derniers caractères. Attention, si vous utilisez FNAM!, le nom proprement dit doit occuper 8 caractères (compléter par des espaces s'il le faut). Cette gestion est différente de celle de BASIC...

Les mots du D.O.S. (seulement si vous utilisez des disquettes).
C'est dans cette zone que viennent se placer les définitions lues sur la disquette système lors du chargement.

La pile des données
128 octets par défaut. La base de la pile est à l'adresse donnée par le mot S0, et l'adresse actuelle de son sommet par le mot SP@.
La pile est en fait "tête en bas" en mémoire, c'est-à- dire que sa base est au-dessus de son sommet, comme le montre l'exemple suivant :

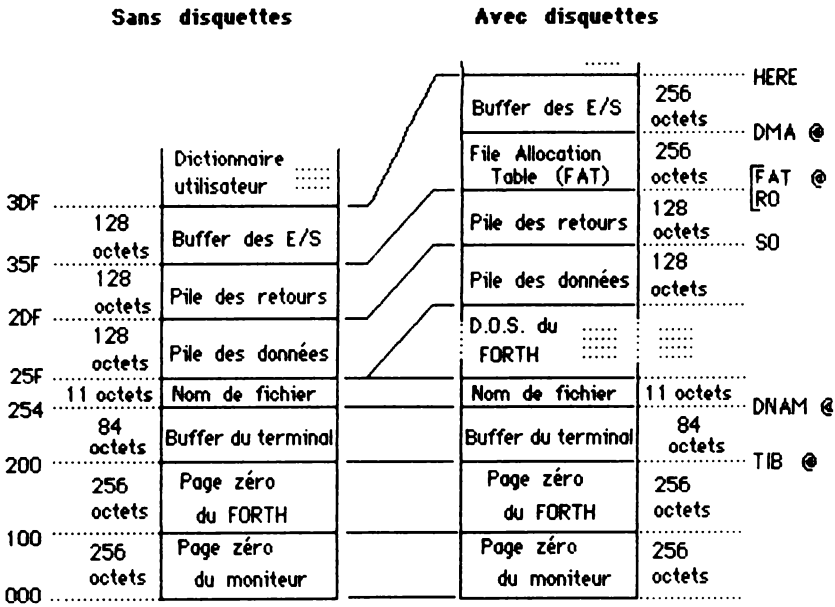
```
SP!  (pour réinitialiser la pile)
HEX S0 . SP  .   62DF 62DF OK
12 23 S0 . SP  .   62DF 62DB OK
```

La pile des retours
128 octets par défaut. Comme la pile des données, elle est "la tête en bas".
La base de la pile (adresse la plus haute) est donnée par RØ, et l'adresse courante du sommet par le mot RP@.

La table d'allocation des secteurs de la disquette (seulement si vous utilisez des disquettes)
Zone de 256 octets. Voir sa description à l'annexe B (mot FAT).

Le buffer des entrées-sorties
Zone de 256 octets par laquelle transitent toutes les informations lues ou à écrire sur le stockage de masse.

L'implantation de ces différentes zones est réalisée comme suit :



Les adresses indiquées à gauche sont relatives au début de la RAM système (2000 pour un MO5, et 6000 pour un TO7 ou un TO7-70). Par exemple sur un MO5, la page zéro FORTH commence en 2100, alors qu'elle commence en 6100 avec un TO7...

Il est possible pour certaines applications d'augmenter les tailles des piles en fixant les valeurs DP, DMA, FAT, RP0, et SP0 puis en réinitialisant le système par COLD...
— DP est le "pointeur dictionnaire", lu par HERE.
— RP0 et SP0 sont les pointeurs de base des piles, lus respectivement par R0 et S0.
Exemple:

```
HEX  S0 80 -        (début de la zone 'pile des données)
DUP A00 + DP!  DUP 900 + FAT !
DUP 800 + DUP DMA RP0 !
400 + SP0 ! COLD
```

vous donnera 1k-octet pour chaque pile, et permute les zones "buffer des entrées-sorties" et "table d'allocation de la disquette". Après cela, vos définitions en RAM sont perdues. Cette manipulation est à faire lors de l'initialisation...

Annexe F

Mise en service

F.1 Mise en service du système

Après vérification de l'ensemble des branchements, mettre la cartouche FORTH dans la trappe, puis allumez dans l'ordre :
1 Le téléviseur
2 Le lecteur de disquette vide
3 Les autres périphériques éventuels
4 L'unité centrale

Vous accédez ainsi au menu sur l'écran.

F.2 Utilisation du clavier

En mode interactif :
Le langage prend en compte *toutes les touches* tapées, y compris celles qui correspondent à des caractères non affichables.
Si par exemple vous tapez les touches suivantes :
[V] [L] [I] [Z] [←] [S] [T]

Forth essaiera d'exécuter l'instruction correspondant à cette séquence de touches... (sept caractères, de codes ASCII respectifs 86, 76, 73, 90, 8, 83 et 84)
Et cela bien que l'écran affiche effectivement "VLIST" !!!
Il va de même pour les touches spéciales suivantes :
[STOP] [RAZ] [INS] [→] [↑] [↓] [ ]

ainsi que pour les autres caractères de contrôle...

Les exceptions sont les touches suivantes :
[ENTREE] Validation du flot d'entrée et passage à la ligne.
[ACC] Caractères accentués (Voir Annexe 1).
[EFF] Suppression du dernier caractère de la ligne, avant validation.
...et leurs équivalents :
[CNT] [M], [CNT] [V]
(respectivement [ENTREE] et [ACC])

Commentaires :
Une séquence du type :
[RAZ] [ENTREE]
efface l'écran, mais provoque un message d'erreur "UK ERROR" (Nom inconnu)...
... A moins que vous n'ayez défini le mot correspondant, en tapant au clavier la séquence de touches suivante :
[:] [ESPACE] [RAZ] [ESPACE] [;] [ENTREE]
Noter que lors de la frappe de la touche RAZ, l'écran s'est effacé...
La définition ne nécessite pas d'action, le simple appui de la touche exécutant l'effacement...
Un VLIST effacera l'écran au passage, lors de l'affichage de ce nom...
Vous pouvez de la même manière définir des mots cachés, qui n'apparaîtront pas dans un VLIST, en prenant par exemple pour nom des séquences telles que :
[A] [←] [←] [←]

Annexe G

Glossaire des mots et abréviations utilisés

adresse
Un nombre non signé (de Ø à 65535) permettant de localiser un octet dans la mémoire de l'ordinateur. Les adresses permettent aussi de localiser les nombres (16 bits en 2 octets, ou 32 bits en 4 octets) et les chaines de caractères (adresse du premier octet de la chaine).

adresse de compilation
En abrégé cfa (code field address). Le cfa d'un mot FORTH XXX est l'adresse du "champ code" du mot.

adresse de lien
En abrégé lfa (link field address). Le lfa d'un mot FORTH XXX est l'adresse du "champ lien" de XXX.

baud
Unité de vitesse de transmission série. "1 baud" signifie "1 bit par seconde". La transmission d'un caractère nécessite une dizaine de bits, donc une communication à 3ØØ bauds transmet environ 3Ø caractères par secondes.

block
L'unité de lecture ou d'écriture des données sur cassette ou sur disquette. Un block comporte 1Ø24 octets. Chaque écran de l'éditeur (1ØØØ caractères) occupe un block.

buffer
ou "zone tampon". Partie de la mémoire où sont rangées des données en vue d'un traitement ultérieur.
En particulier, les échanges entre l'ordinateur et son environnement transitent par des buffers.
FORTH utilise aussi des "buffers de block" (voir ci-dessous), où sont rangés entre autres les écrans de l'éditeur.

buffer de block
Les buffers de blocks, de 1Ø24 octets, sont utilisés pour les échanges avec le magnétophone ou les disquettes. En l'absence d'autre précision, le mot "buffer" désigne les buffers de blocks.

buffer d'entrée du terminal
En abrégé tib (pour "terminal input buffer"). Buffer de 8Ø octets, qui contient les données en provenance du périphérique considéré en entrée (en général le clavier, mais FORTH permet de positionner en entrée tout périphérique capable d'émettre).

cf cfa
Voir "champ code"

chaîne de caractères
Suite d'octets contenant les codes ASCII de caractères, et repérée en mémoire par l'adresse de son premier octet, et par sa longueur.

champ code
En abrégé cf (code field). Partie de la définition d'un mot du dictionnaire qui contient l'adresse du code machine à exécuter lorsque ce mot est appelé.
Abréviation pour "adresse du champ code" : cfa (code field address).

champ lien
En abrégé lf (link field). Partie de la définition d'un mot du dictionnaire qui contient l'adresse du "champ nom" du mot précédent du dictionnaire (plus précisément du même vocabulaire).
Abréviation pour "adresse du champ lien" : lfa (link field address).

champ nom
En abrégé nf (name field). Partie de la définition d'un mot qui contient le nom par l'intermédiaire duquel le mot est utilisé. Le champ nom est composé d'un octet indiquant entre autres la longueur du nom (et aussi si le mot considéré est immédiat), suivi des caractères composant le nom. Tous les octets correspondant aux caractères du nom ont leur bit 7 à zéro, sauf le dernier, qui indique la fin du nf. Sur le TO-7, cela se traduit par l'impossibilité d'utiliser des minuscules accentuées ou des caractères graphiques (bit 7 à un) pour les noms des mots FORTH.

champ paramètre
En abrégé pf (parameter field). Partie de la définition d'un mot FORTH qui contient les paramètres utilisés par le code machine exécutable (voir "champ code"). Ces paramètres peuvent être des cfa d'autres mots, des nombres, ou des caractères, suivant le "mot de définition" utilisé.

code machine langage machine
C'est le seul langage compris par l'ordinateur. Une routine écrite en code machine est directement exécutable en dehors du FORTH. Tous les langages sont écrits en langage machine, le "noyau FORTH" comme les autres.
On peut aussi définir des mots FORTH en langage machine (leur cfa doit simplement contenir l'adresse d'exécution du programme machine considéré), et on peut appeler du FORTH une routine extérieure au langage (par le mot CALL).

communication série ou parallèle
Voir transmission.

code FORTH
C'est le résultat de la compilation d'un texte source, composé de mots FORTH. Le code FORTH est utilisable directement par l'interpréteur interne.
Le dictionnaire est constitué de code FORTH.

compilation
L'action de lire l'un après l'autre les mots du flot d'entrée et de déposer leur cfa à la fin du dictionnaire.
Cette définition correspond à ce qui s'appelle couramment le mode compilation (positionné par exemple par le mot :).
On utilise aussi parfois le mot "compiler" pour parler de l'action "mettre au dictionnaire", sans qu'il s'agisse nécessairement du cfa d'un mot (par exemple : "compiler en tant que littéral". Voir "littéral").

cond
Voir "valeur de vérité".

contrôleur de communication
Interface de communication, adaptable à l'arrière de votre ordinateur, et pouvant réaliser des transmissions en série (RS-232) ou en parallèle.

définition d'un mot
Ne pas confondre avec "mot de définition".
La définition d'un mot est une procédure exécutable par FORTH, et qui porte un nom. L'ensemble des définitions constitue le dictionnaire.
Classiquement, une définition comporte deux parties: un en-tête, comportant le champ-nom et le champ-lien (utilisé pour la recherche-dictionnaire), et la partie exécutable proprement dite, composée du champ-code et du champ-paramètres.

dictionnaire
Liste des définitions des mots FORTH, chaînées par l'intermédiaire de leur champ-lien. Le dictionnaire est organisé en "vocabulaires". Il est extensible, grâce aux mots de définition.

écran
Ce mot désignera parfois l'écran de votre téléviseur ou de votre moniteur, mais le plus souvent il désignera un "écran de l'éditeur" (en anglais screen, abrégé en scr).
Il s'agit dans ce cas d'un ensemble de données (1Ø24 caractères) contenant le plus souvent du "texte source", mais pouvant être utilisé pour ranger des données.
En standard, un écran comporte 16 lignes de 64 caractères (1Ø24 caractères).
Sur votre appareil, un écran comporte 25 lignes de 4Ø caractères (1ØØØ caractères).
Du point de vue du système, un écran est rangé dans un block, et porte le numéro du block qui le contient. On peut donc le manipuler par tous les mots qui concernent les blocks (en mémoire, un écran occupera donc un buffer de block).

entrées-sorties
Echanges d'informations entre l'ordinateur et son environnement (ses périphériques). Les entrées se font d'un périphérique vers l'unité centrale, les sorties de l'unité centrale vers les périphériques.

flot d'entrée
Suite de caractères reçue par l'ordinateur, destinée à être traitée par l'interpréteur.
En principe, le flot d'entrée provient soit du clavier (par l'intermédiaire du "buffer d'entrée"), soit d'un block (par l'intermédiaire d'un buffer de block) qui peut être en mémoire, sur cassette-magnétophone, ou sur disquette. Ceci est paramétrable, de sorte que le flot d'entrée peut provenir de tout périphérique capable d'émettre.

immédiat
Un mot immédiat est un mot qui s'exécute "immédiatement" lorsqu'il est rencontré pendant une compilation. Les mots immédiats servent en général à modifier le processus habituel de la compilation. Les mots définissant des structures de contrôle doivent être immédiats, mais aussi les mots servant à définir des "littéraux", et plus généralement les mots qui doivent avoir une action différente suivant le mode (compilation ou exécution).

interface
Système électronique permettant l'échange d'informations entre l'unité centrale et les périphériques. Les interfaces comportent souvent une ROM, adressable en totalité ou en partie dans l'espace réservé aux entrées-sorties.

interpréteur externe
Ou "interpréteur de texte". Ensemble des mots (le mot final est INTERPRET) qui répètent sans cesse les actions suivantes :
- Lire le mot suivant du flot d'entrée.
- Le chercher au dictionnaire.
- L'exécuter ou le compiler, suivant le mode.

Si un mot n'est pas trouvé au dictionnaire, l'interpréteur essaie de le convertir en nombre (compte tenu de la base).
Si cela est possible (c'est bien un nombre), il sera déposé sur la pile ou compilé comme littéral, suivant le mode.
Si cela est impossible, l'opération s'achève par un traitement de l'erreur (entre autres, message "UNKNOWN").

interpréteur interne
Ou "interpréteur d'adresses". C'est le coeur du système, son rôle est d'interpréter des séquences d'adresses de compilation (régler le séquencement des opérations) et d'exécuter les mots correspondants. C'est lui, en particulier, qui exécute un mot donné par son cfa.

lf lfa
Voir "champ lien"

littéral
Un littéral est un nombre ou une chaîne de caractères à utiliser "tel quel" par l'ordinateur, sans interprétation. Lorsqu'on utilise un littéral à l'intérieur d'une définition, il est automatiquement précédé d'un opérateur spécifique (suivant le type de littéral), pour que l'interpréteur ne le confonde pas avec un cfa.

mot de définition

Un mot de définition est un mot qui, lorsqu'on l'exécute, crée un nouveau mot dans le dictionnaire. Le nom de ce nouveau mot est pris dans le flot d'entrée.

Les mots de définition les plus courants sont VARIABLE, CONSTANT, et surtout le mot ":"

nf nfa

Voir "champ nom"

nombre

FORTH ne travaille de base qu'en nombres entiers, signés ou non. Suivant le type de nombre, l'occupation en mémoire est la suivante pour le Standard Forth-79:

type	valeurs possibles	nombre de bits minimum
bit	0 ou 1	1
caractère	0 ... 127	7
octet	0 ... 255	8
nombre		
signé	-32768 ... 32767	16
non signé	0 ... 65535	16
nombre double longueur		
signé	-2147483648 ... 2147483647	32
non signé	0 ... 4294967295	32

Les nombres en double longueur son représentés en mémoire et sur la pile comme deux nombres 16 bits. Sur la pile, les 16 bits de poids fort (y compris le bit de signe) d'un nombre double longueur sont "au dessus" de la pile (en fait en dessous, car la pile est en réalité implantée de haut en bas). En mémoire, ils occupent les deux octets d'adresses les plus basses.

Spécificités du FORTH du TO-7

— Les nombres 16 bits sont rangés en mémoire et sur la pile dans l'ordre des octets poids fort-poids faible.

— Les caractères affichables par le TO-7 correspondent aux codes ASCII de 0 à 255 (128 à 153 pour les caractères minuscules accentuées, et 154 à 255 pour les caractères graphiques redéfinissables). Mais les noms des mots FORTH (les noms du dictionnaire) ne peuvent contenir que des caractères dont le code ASCII est inférieur ou égal à 127 (pas de minuscules accentuées ni de caractères graphiques).

périphérique
Appareil permettant à l'unité centrale de communiquer avec son environnement.
Les principaux types de périphériques sont :
— En entrée (vers l'ordinateur) : clavier, crayon optique, manettes de jeux...
— En sortie (vers l'extérieur) : écran, imprimante, table traçante,...
— En entrée-sortie : stockage de masse (cassettes, disquettes, autre ordinateur,...)

pf pfa
Voir "champ paramètre"

pile des données
(Parameter Stack). Pile (dernier entré, premier sorti) contenant des nombres 16 bits. Ces nombres peuvent représenter des adresses, des nombres, des caractères, des valeurs de vérité, etc. Cette pile est utilisée par de nombreux mots FORTH, pour obtenir des paramètres ou ranger des résultats.
Sauf indication contraire, le mot "pile" désigne la pile des données.

pile des retours
(Return Stack). Pile (dernier entré, premier sorti) contenant les adresses de retour aux mots en cours d'exécution.
Lorsqu'un mot X appelle un mot Y, l'adresse du mot qui suit Y dans la définition de X (adresse de retour à X après exécution de Y) est placée sur la pile des retours.
La pile des retours peut aussi être utilisée différemment, moyennant précautions. Par exemple, pour les paramètres de contrôle de boucle, ou pour sauvegarder temporairement certaines valeurs (qu'il ne faut pas oublier d'enlever avant que le système ait à utiliser la pile des retours).

port de communication
Dans ce manuel, ce terme désigne la partie "transmission série RS-232" du contrôleur de communication adaptable à l'arrière de votre appareil.

RS-232
Standard de transmission série. Voir transmission.

stockage de masse
ou "mémoire externe". Périphérique en entrée-sortie permettant la sauvegarde d'informations lorsque le courant est coupé. Le stockage de masse consiste le plus souvent en bandes ou cassettes magnétiques, en disquettes ou disque dur.

source
Ou "texte source", ou "programme source".
Suite de caractères composant des mots connus du FORTH, et utilisable par l'interpréteur externe.
Le texte source constitue le flot d'entrée. Il peut provenir du clavier, mais est le plus souvent organisé en écrans et sauvegardé sur cassette ou sur disquette.

tib
Voir "buffer d'entrée du terminal".

transmission (série ou parallèle)
Les informations qui transitent entre l'unité centrale et les périphériques sont le plus souvent composées d'octets, ou de caractères, comportant 7 ou 8 bits. Ces bits peuvent être transmis :
— En parallèle (8 bits simultanément). Exemple : imprimante parallèle.
— En série (1 bit à la fois). Dans ce cas, il faut préciser le nombre de bits à prendre en compte (7 ou 8), et la vitesse de transmission (11Ø à 96ØØ bauds)

transportabilité
Ce terme indique que les mêmes mots, définis à partir du vocabulaire de base, donneront un résultat identique si on les exécute sur tout autre ordinateur travaillant en FORTH-79 STANDARD.
Pour être transportable, un mot doit être écrit en n'utilisant que des mots "standard". Il convient en particulier d'éviter les mots propres au système (par exemple pour le TO-7, les mots ayant trait au graphique et à la musique).

valeur de vérité
En abrégé cond (flag en anglais). Un nombre qui peut être dans deux "états logiques", nul ("faux") et non nul ("vrai"), et qui sert à traduire si une condition est réalisée ou non.
Les mots standards du FORTH qui laissent sur la pile une valeur de vérité laissent 1 pour "vrai", et Ø pour "faux".

variable "user"
En principe utilisées dans les systèmes multi-utilisateurs, pour la réentrance des mots du FORTH. Une copie des variables du système est alors conservée dans une zone spéciale, dite "zone user".
Dans le cas particulier du TO-7, toutes les variables du système doivent être de ce type, puisque le dictionnaire de base est en cartouche ROM, donc ses variables ne seraient pas modifiables...

vocabulaire
Liste de définitions de mots du dictionnaire, chainées entre elles, et formant un "sous-dictionnaire". L'emploi de vocabulaire offre les avantages d'une plus grande rapidité dans la recherche-dictionnaire, de la possibilité d'avoir plusieurs définitions portant le même nom, et une plus grande lisibilité de la liste des noms du dictionnaire (VLIST).
FORTH est un vocabulaire, c'est le vocabulaire de base. L'éditeur est un autre vocabulaire (vocabulaire EDITOR).

Annexe H

Index des mots du FORTH

Les nombres entre parenthèses renvoient aux chapitres et annexes où sont décrits les mots.

MOTS	CHAPITRES OU ANNEXES
!	(8)
!CSP	(4)
''	(5)
''''	(5)
#	(5)
# >	(5)
#BUF	(10)
#S	(5)
$!	(5)
$+	(5)
$<	(5)
$=	(5)
'	(8)
(	(11)
(')	(5)
(+LOOP)	(3)
(.')	(9)
(/LOOP)	(3)
(;CODE)	(4)
(ABORT)	(11)
(BUFFER)	(B)
(CR)	(9)
(DEMIT)	(B)
(DIO)	(B)
(DKEY)	(B)
(DO)	(3)
(EMIT)	(9)
(FIND)	(8)
(FLUSH)	(B)
(FR/W)	(10)
(GODO)	(3)
(GPRT)	(9)
(INTRP)	(4)
(KEY)	(9)

(KBUF)..... (10)
(KEMIT)..... (10)
(KFLUSH)..... (10)
(KIO)..... (10)
(KKEY)..... (10)
(LINE)..... (11)
(LOOP)..... (3)
(NBR)..... (5)
(OF)..... (3)
(PCR)..... (9)
(PEMIT)..... (9)
(PIO)..... (10)
(PKEY)..... (9)
*..... (2)
*/..... (2)
*/MOD..... (2)
+..... (2)
+!..... (8)
+-..... (2)
+BUF..... (10)
+LOOP..... (3)
,..... (4)
-..... (2)
-FIND..... (8)
-TRAILING..... (5)
..... (9)
."..... (9)
.#..... (7)
.LINE..... (11)
.R..... (9)
.S..... (11)
.b..... (7)
/..... (2)
/LOOP..... (3)
/MOD..... (2)
0..... (10)
0<..... (2)
0=..... (2)
0>..... (2)
0BRANCH..... (3)
1..... (1)
1+..... (2)
1+!..... (8)
1-..... (2)
1-!..... (8)
2..... (1)
2*..... (2)
2+..... (2)
2-..... (2)
2/..... (2)
3..... (1)
79-STANDARD... (11)
:..... (4)
;..... (4)
;S..... (4)
<..... (2)
<#..... (5)
<CMOVE..... (8)
=..... (2)
>..... (2)
><..... (1)
>IN..... (9)
>MUS..... (7)
>MUS+..... (7)
>OUT..... (9)
>R..... (1)
?..... (8)
?COMP..... (4)
?CSP..... (4)
?DUP..... (1)
?ERROR..... (11)
?EXEC..... (4)
?FIND..... (8)
?PAIRS..... (3)
?S..... (4)
?STOP..... (9)
?TERMINAL..... (9)
?WORD..... (5)
@..... (8)
ABORT..... (11)
ABS..... (2)
AGAIN..... (3)
ALLOT..... (4)
AND..... (2)
ARRAY..... (4)
ARRAY$..... (4, 5)
ASC..... (5)
AUTORUN..... (11)
B/BUF..... (1)
B/S..... (B)

BACK..... (3)
BACKUP..... (B)
BANK..... (8)
BASE..... (5, 9)
BBLOCK..... (1)
BEGIN..... (3)
BELL..... (11)
BL..... (9)
BLACK..... (6)
BLANKS..... (8)
BLK..... (10)
BLOCK..... (10)
BLUE..... (6)
BOX..... (6)
BOXF..... (6)
BOXFTO..... (6)
BOXPEN..... (6)
BOXTO..... (6)
BRANCH..... (3)
BS..... (9)
BUFFER..... (10)
BUFFERS..... (10)
BUTTON..... (9)
C!..... (8)
C,..... (4)
C/L..... (11)
C@..... (8)
CALL..... (11)
CARRAY..... (4)
CASE..... (3)
CASS..... (B)
CFA..... (8)
CHARTAB..... (5)
CHR$..... (5)
CLS..... (6)
CMOVE..... (8)
COLD..... (11)
COLOR..... (6)
COLOROFF..... (6)
COLORON..... (6)
COMPILE..... (4)
CONSOLE..... (9)
CONSTANT..... (4)
CONTEXT..... (4, 8)
CONVERT..... (5)
COPY-BLOCKS.. (B)
COUNT..... (5)
CR..... (9)
CREATE..... (4)
CSP..... (4),
CSRLIN..... (6)
CSROFF..... (6)
CSRON..... (6)
CTABLE..... (4)
CURRENT..... (4)
CYAN..... (6)
D!..... (8)
D*..... (2)
D+..... (2)
D+-..... (2)
D-..... (2)
D-ERR..... (10)
D..... (9)
D.R..... (9)
DØ=..... (2)
D<..... (2)
D=..... (2)
D@..... (8)
DABS..... (2)
DARK..... (6)
DCONSTANT..... (4)
DDROP..... (1)
DDUP..... (1)
DECIMAL..... (5, 9)
DEFGR..... (5)
DEFINITIONS.... (4)
DENSITY..... (B)
DEPTH..... (1)
DIGIT..... (5)
DIR..... (B)
DISK..... (B)
DLITERAL..... (4)
DMA..... (1)
DMAX..... (2)
DMIN..... (2)
DN*..... (2)
DNAM..... (10)
DNEGATE..... (2)
DO..... (3)
DOES>..... (4)

DOS..... (B)
DOVER..... (1)
DP..... (8)
DPL..... (9)
DRIVE..... (B)
DROP..... (1)
DROT..... (1)
DSWAP..... (1)
DU< ..... (2)
DUMP..... (8)
DUP..... (1)
DUR..... (7)
DVARIABLE..... (4)
EDIT..... (11, A)
EDITOR..... (4, 8, A)
ELSE..... (3)
EMIT..... (9)
EMPTY-BUFFERS (10)
ENCLOSE..... (5)
ENDCASE..... (3)
ENDOF..... (3)
EOF..... (10)
ERASE..... (8)
ERR..... (11)
ERROR..... (11)
EXECUTE..... (4)
EXIT..... (3)
EXPECT..... (9)
FASC!..... (10)
FAT..... (B)
FENCE..... (8)
FILL..... (8)
FIND..... (8)
FIRST..... (8, 10)
FNAM!..... (10)
FNB!..... (10)
FORGET..... (8)
FORMAT..... (B)
FORTH..... (4, 8)
FRAME..... (6)
FTYP!..... (10)
GCOLOR (6)
GETC..... (10)
GLOBAL..... (6)
GO..... (11)
GODO..... (3)
GREEN..... (6)
HERE..... (8)
HEX..... (5, 9)
HILO..... (1)
HLD..... (5)
HOLD..... (5)
HOME..... (6)
I..... (3)
ID..... (8)
IF..... (3)
IMMEDIATE..... (4)
IN?..... (9)
INITMUS..... (7)
INITSCR..... (6)
INK..... (6)
INKEY..... (9)
INPEN..... (9)
INPUT..... (9)
INPUT$..... (9)
INPUTPEN..... (9)
INSTR..... (5)
INTERPRET..... (4)
INVCOLOR..... (6)
IO..... (10)
IOERR..... (10)
J..... (3)
KEY..... (9)
KILL..... (B)
L/SCR..... (11)
LATEST..... (8)
LEAVE..... (3)
LEFT$..... (5)
LEN..... .(5)
LFA..... .(8)
LIMIT..... (8)
LINE..... (6)
LINETO..... (6)
LINPUT..... (9)
LINPUT$..... (9)
LIST..... (11)
LIT..... (4)
LITERAL..... (4)
LOAD..... (10)
LOCAL..... (6)

LOCATE..... (6)
LOOP..... (3)
M*..... (2)
M/..... (2)
M/MOD..... (2)
MAGENTA..... (6)
MASK..... (6)
MAX..... (2)
MAXBUF..... (10)
MESSAGE..... (11)
MID$..... (5)
MIN..... (2)
MLEN..... (5)
MOD..... (2)
MOTOROFF..... (10)
MOTORON..... (10)
MOVE..... (8)
MUSBUF..... (7)
MUSTAB..... (7)
NEGATE..... (2)
NFA..... (8)
NOMASK..... (6)
NOT..... (2)
NOTE..... (7)
NUMBER..... (5)
OCT..... (7)
OF..... (3)
OFFSET..... (10)
ONERR..... (11)
ONPEN..... (9)
OR..... (2)
OVER..... (1)
PAD..... (5)
PAGE..... (6)
PALE..... (6)
PAPER..... (6)
PEN..... (6)
PFA..... (8)
PICK..... (1)
PITCH..... (7)
PLAY..... (7)
POINT..... (6)
POS..... (6)
PREV..... (10)
PRINTER..... (9)
PSET..... (6)
PTRIG..... (9)
PUTC..... (10)
QUERY..... (9)
QUIT..... (11)
R/W..... (10)
RØ..... (1)
R>..... (1)
R@..... (1)
RED..... (6)
REDIT..... (11, A)
REGD..... (11)
REGU..... (11)
REGX..... (11)
REGY..... (11)
RENUM..... (B)
REPEAT (3)
RIGHT$..... (5)
RND..... (11)
ROLL..... (1)
ROT..... (1)
RP!..... (1)
RPØ..... (1)
RP@..... (1)
RS-B..... (1)
RS-S..... (10)
RUN..... (11)
S->D..... (2)
SØ..... (1)
SAVE..... (11)
SAVE-BUFFERS.. (10)
SCR..... (11)
SCREEN..... (6)
SCRPRT..... (9)
SCROLL..... (6)
SEC-RD..... (B)
SEC-WR..... (B)
SIGN..... (5)
SIZE..... (6)
SKIP..... (10)
SKIPB..... (10)
SMUDGE..... (4)
SP!..... (1)
SPØ..... (1)
SP@..... (1)

SPACE ..... (9)
SPACES ..... (9)
SQUISH ..... (1)
STATE ..... (4)
STICK ..... (9)
STR$ ..... (5)
STRING ..... (4, 5)
SUB$! ..... (5)
SWAP ..... (1)
SWAPDROP ..... (1)
SYSVEC ..... (11)
TABLE ..... (4)
TEMP ..... (7)
TEXT ..... (5)
THEN ..... (3)
THRU ..... (10)
TIB ..... (9)
TOGGLE ..... (8)
TRAVERSE ..... (8)
TYPE ..... (9)
U* ..... (2)
U. ..... (9)
U.R ..... (9)
U/MOD ..... (2)
U< ..... (2)
UD ..... (9)
UD.R ..... (9)
UDN* ..... (2)
UNTIL ..... (3)
UPDATE ..... (10)
USE ..... (10)
USTR$ ..... (5)
VAL ..... (5)
VARIABLE ..... (4)
VLIST ..... (8)
VOC-LINK ..... (4)
VOCABULARY ... (4)
WAIT ..... (11)
WHERE ..... (11)
WHILE ..... (3)
WHITE ..... (6)
WIDTH ..... (8)
WINDOW ..... (6)
WORD ..... (5)
XOR ..... (2)
YELLOW ..... (6)
[..... (4)
[COMPILE] ..... (4)
] ..... (4)
do ..... (7)
do+ ..... (7)
dur ..... (7)
fa ..... (7)
la ..... (7)
mi ..... (7)
oct ..... (7)
pause ..... (7)
pitch ..... (7)
re ..... (7)
si ..... (7)
sol ..... (7)
temp ..... (7)

Annexe I

Codes ASCII et touches du clavier (suivant l'ordinateur)

Décimal	Hexa	Touche(s) à appuyer			Action éventuelle sur l'affichage (voir EMIT)
Caractères de contrôle					
0	0				
1	1	CNT-A	donne le code 91		
2	2	STOP	ou	CNT-B	
3	3			CNT-C	
4	4			CNT-D	
5	5			CNT-E	
6	6			CNT-F	
7	7			CNT-G	Sonnette
8	8	←	ou	CNT H	Recul du curseur (d'un caractère)
9	9	→	ou	CNT-I	Avance du curseur
10	A	↓	ou	CNT-J	Saut de ligne vers le bas
11	B	↑	ou	CNT-K	Saut de ligne vers le haut
12	C	RAZ	ou	CNT-L	Effacement de l'écran
13	D	ENTREE	ou	CNT-M	Passage à la ligne
14	E			CNT-N	Mode semi-graphique (TELETEL) *
15	F			CNT-O	Mode alphanumérique
16	10			CNT-P	(Fin TELETEL) *
17	11			CNT-Q	Curseur visible
18	12			CNT-R	Répétition du dernier caractère *
					Envoi de 65, 18, 4 : "AAAAA OK"

* Sur TO7 seulement

19	13		CNT - S	
20	14		CNT - T	Curseur invisible
21	15		CNT - U	
22	16	ACC ou	CNT - V	Voir les caractères spéciaux
23	17		CNT - W	
24	18		CNT - X	Efface la fin de ligne
25	19		CNT - Y	
26	1A	CNT - Z donne le code 93		
27	1B			Début de séquence d'attribution de couleur...
28	1C	INS		
29	1D	EFF		Effacement d'un caractère
30	1E	⮌		Curseur en début d'écran
31	1F			Début de séquence de positionnement du curseur ou de format d'écran.

Caractères affichables

32	20	Barre d'espacement	Espace
33	21	Maj-!	!
34	22	Maj-''	''
35	23	Maj-#	#
36	24	Maj.-$	$
37	25	Maj-%	%
38	26	Maj-&	&
39	27	Maj-'	'
40	28	Maj-(	(
41	29	Maj-)	)
42	2A	*	*
43	2B	+	+
44	2C	,	,
45	2D	-	-
46	2E	.	.
47	2F	/	/
48	30	0	0
49	31	1	1
50	32	2	2
51	33	3	3
52	34	4	4
53	35	5	5
54	36	6	6
55	37	7	7
56	38	8	8
57	39	9	9
58	3A	Maj-:	:
59	3B	Maj-;	;

60	3C	Maj-<	<
61	3D	Maj-=	=
62	3E	Maj->	>
63	3F	Maj-?	?
64	40	@	@

En mode majuscules uniquement

65	41	A	A
66	42	B	B
67	43	C	C
68	44	D	D
69	45	E	E
70	46	F	F
71	47	G	G
72	48	H	H
73	49	I	I
74	4A	J	J
75	4B	K	K
76	4C	L	L
77	4D	M	M
78	4E	N	N
79	4F	O	O
80	50	P	P
81	51	Q	Q
82	52	R	R
83	53	S	S
84	54	T	T
85	55	U	U
86	56	V	V
87	57	W	W
88	58	X	X
89	59	Y	Y
90	5A	Z	Z
91	5B	CNT - A	[
92	5C		
93	5D	CNT - Z	]
94	5E	Maj-@	Flèche vers le haut
95	5F		□
96	60		□

En mode minuscules uniquement

97	61	a	a
98	62	b	b
99	63	c	c
100	64	d	d
101	65	e	e
102	66	f	f
103	67	g	g
104	68	h	h
105	69	i	i
106	6A	j	j
107	6B	k	k
108	6C	l	l
109	6D	m	m
110	6E	n	n
111	6F	o	o
112	70	p	p
113	71	q	q
114	72	r	r
115	73	s	s
116	74	t	t
117	75	u	u
118	76	v	v
119	77	w	w
120	78	x	x
121	79	y	y
122	7A	z	z
123	7B		{
124	7C		□
125	7D		}
126	7E		□
127	7F		□

Caractères spéciaux

En mode minuscules uniquement

128	80	ACC + c + c	ç
129	81	ACC + Ø + a	à
130	82	ACC + Maj-' + a	á
131	83	ACC + Maj-@ + a	â
132	84	ACC + Maj-" + a	ä
133	85	ACC + Ø + e	è
134	86	ACC + Maj-' + e	é
135	87	ACC + Maj-@ + e	ê
136	88	ACC + Maj-" + e	ë
137	89	ACC + Ø + i	ì
138	8A	ACC + Maj-' + i	í
139	8B	ACC + Maj-@ + i	î
140	8C	ACC + Maj-" + i	ï
141	8D	ACC + Ø + o	ò
142	8E	ACC + Maj-' + o	ó
143	8F	ACC + Maj-@ + o	ô
144	90	ACC + Maj-" + o	ö
145	91	ACC + Ø + u	ù
146	92	ACC + Maj-' + u	ú
147	93	ACC + Maj-@ + u	û
148	94	ACC + Maj-" + u	ü
149	95	ACC + Ø + n	ǹ
150	96	ACC + Maj-' + n	ń
151	97	ACC + Maj-@ + n	ñ
152	98	ACC + Maj-" + n	n̈
153	99	ACC + {	ß
154	9A		caractères graphiques définis par l'utilisateur
...	...		
255	FF		

Messages d'erreur commentés

BA ERROR (Bad argument)
Réservé pour une utilisation future.

CD ERROR (Conditionals not paired)
Signale que lors de la dernière compilation, FORTH a rencontré un mot de fin de structure conditionnelle ou répétitive, mais n'a pas trouvé de mot de début de structure.
Exemples :
ELSE sans IF
THEN sans IF ou GODO
LOOP sans DO
UNTIL sans BEGIN ...
Signale aussi un mauvais appariement des mots de structuration. Par exemple une structure du genre DO...UNTIL, qui n'existe pas en FORTH. Oubliez la dernière définition du dictionnaire, et refaites-la.

CO ERROR (Compilation only)
Le mot ne peut être employé qu'en mode compilation, dans une définition par :...; C'est le cas pour tous les mots de structuration, comme IF, THEN, DO, etc.

DF ERROR (Dictionary Full)
Il n'y a plus de place pour créer un nouveau mot au dictionnaire. Peut-être avez-vous utilisé trop de buffers. Dans ce cas, oubliez le nom en cours de définition, sauvegardez vos écrans, diminuez le nombre de buffers, et recompilez votre application.
Le nombre de buffers minimum est 2.

DL ERROR (Delimiter not found)
Le flot d'entrée ne contient pas le délimiteur cherché.

DR ERROR (Disk range)
Concerne la gestion des disquettes en blocs physiques. Le numéro de bloc ne correspond pas à un lecteur de disquettes présent (Attention à la base utilisée pour les nombres).

ES ERROR (Empty Stack)
La pile des données est vide.
Si une erreur de ce type survient lors de l'exécution d'un mot, la vérification ne se fait qu'à postériori, à la fin du mot. Ce qui signifie que le mot s'exécute d'abord, utilisant les données qui sont sous la pile.

FS ERROR **(Full stack)**
La pile des données est saturée (contient plus de 64 nombres).
Si une erreur de ce type provient lors de l'exécution d'un mot, la vérification n'a lieu qu'à postériori. Votre mot continuera donc d'abord à s'exécuter, empilant éventuellement des données hors de la pile, dans la zone des variables système.
Les résultats peuvent être extrêmement curieux...

I> ERROR **(Input > 255 characters)**
Vous avez essayé de définir une chaîne de caractères de longueur supérieure à 255.
Ce message concerne les chaînes définies par '' ou .'', mais aussi les remarques entre parenthèses. Il vous manque probablement un guillemet ou une parenthèse fermante pour terminer une chaîne de caractères ou une remarque.

IO ERROR
Erreur d'entrée-sortie. Ce message concerne toutes les communications entre l'ordinateur et l'imprimante, les cassettes, ou les disquettes. Il peut indiquer aussi bien une erreur en lecture qu'en écriture.

LO ERROR **(Load only)**
Réservé pour une utilisation ultérieure.

NU ERROR **(Not unique)**
Le nom du dernier mot défini figure déjà au dictionnaire. Ce message n'est qu'un avertissement. Les mots déjà compilés en utilisant la version précédente ne sont pas affectés, mais seule la dernière version sera prise en considération pour les définitions ultérieures, tant que vous ne l'aurez pas « oubliée ».

PR ERROR **(Protected word)**
Le mot que vous essayez d'oublier est protégé. Il fait partie du dictionnaire de base (en cartouche ROM), ou a été sauvegardé sur cassette et rechargé par le mot RUN. Dans ce cas, vous pouvez le déprotéger en agissant sur la variable FENCE (Par exemple : Ø FENCE !).

TB ERROR **(Too big)**
Réservé pour une utilisation future.

UF ERROR **(Unfinished structure)**
Lors de la dernière compilation, une structure conditionnelle ou répétitive n'a pas été terminée. Par exemple DO sans LOOP, IF sans THEN, etc. Oubliez la dernière définition du dictionnaire, et refaites la.

UK ERROR (UnKnown name)
Le mot ne figure pas dans le dictionnaire (dans son état actuel, c'est-à-dire dans le vocabulaire actuellement utilisé). Peut-être l'avez vous mal orthographié... D'autre part, attention aux caractères de contrôle, invisibles à l'écran mais interprétés par FORTH. Exemples : En mode direct, utilisation de la flèche à droite au lieu de l'espace, de la flèche à gauche au lieu de la touche EFF, de la touche RAZ pour effacer l'écran, etc. Ou encore, rebond sur la touche STOP à la sortie de l'éditeur...

VF ERROR (Vocabulary full)
Il n'est pas possible de créer un vocabulaire supplémentaire. Vous ne pouvez avoir en mémoire plus de 4 vocabulaires simultanément, FORTH et EDITOR inclus.

XO ERROR (Execution only)
Signale que le mot ne peut être employé qu'en dehors d'une définition par :...;

Index

!, 120
!CSP, 51
", (chaîne), 75
', 75
›, 77
Buf, 161
$!, 77
$+, 78
$‹, 77
(“), 79
(, 179
(."), 139
(ABORT), 180
(BUFFER), 207
(CR), 139
(DEMIT), 207
(DIO), 208
(DKEY), 208
(EMIT), 139
(FIND), 121
(FLUSH), 209
(FR/W), 161
(GRPRT), 140
(KBUF), 162
(KEMIT), 162
(KEY), 140
(KFLUSH), 162
(KIO), 163
(KKEY), 163
(LINE), 180
(NBR), 77
(PCR), 141
(PEMIT), 141
(PIO), 164
(PKEY), 142
*, 22
*/, 22
*/MOD, 22
+!, 121
+, 23
+-, 23
+BUF, 164
+LOOP, 36
,, 120
,, 52
-, 23
-FIND, 121
-TRAILING, 78
.", 142
., 110
.; 142
.B, 110
.LINE, 180
.R, 142
.S, 181
/, 23
/LOOP, 37
/MOD, 24
0‹, 24
0=, 24
0›, 24
0BRANCH, 38
1+!, 122
1-!, 122
79-standard (mot), 181
:, 52
;CODE, 51
;S, 53
‹, 80
‹, 25
‹CMOVE, 122
=, 25
›, 26
›IN, 142
›MUS+, 111
›MUS, 110
›OUT, 142
›R, pile des retours, 11
?, 123
?CSP, 54
?COMP, 53
?EXEC, 54
?FIND, 123
?PAIRS, 38
?S, 54
?STOP, 144
?TERMINAL, 144
?WORD, 80
?ERROR, 181
', 124
[, 70
[Compile], 70
], 72

A

ASCII, 74, 83, 106, 139-140, 141, 147, 151, 182, 259-263
ABORT, 182
ABS, 26
ADR, notation, 10
Adresse de compilation, 243
Adresse de lien, 243
Adresse, 243
Affichage, 89, 137, 145, 156, 180-181
  (avec .), 142
  de catalogue, 212
  de la mémoire, 127-128
AGAIN, 38
ALLOT, 55
AND, 26
Apostrophe, 71
Appels de sous-programme, 179-190
Array$, 56
Array$, 80
Array, 55
ASC, 81
Autorun, 182

B

BUFF, 164
B/S, 209
BACK, 38
BACKUP, 209

BANK, 124
Base dix, 83
Base seize, 148
Base seize, 86
Base, 81, 85, 144
Baud, 243
BBLOCK, 165
BEGIN, 39
Bell, 182
Bit-map, 21
BL, 144
BLACK, 95
BLANKS, 124
BLUE, 95
BLK, 165
Block, 244
Block, mot, 166
Blocks (stockage), 159
Blocks, 205
Boucle indicée, 41-42
Boucles sans compteur, 39
Boucles, 35-36, 41-46
BOX, 95
BOXF, 95
BOXFTO, 96
BOXPEN, 96
Branch (mot), 35
Branch, 41
Branchements inconditionnels (BRANCH), 41
BS, 145
Buffer, 159, 244
  d'entrée du terminal, 155, 251
  de block, 244
  des entrées au clavier, 238
  des entrées-sorties, 239
  mots, 167
BUTTON, 145

C

C (notation pour caractère), 10, 56
C!, 124
C/L, 183
C, 124
Calculs arithmétiques et logiques, 19-34
CALL, 183
Caractères accentués, 140, 141, 151
Caractères graphiques, 82, 84, 89
Caractères, taille de, 107
CARRAY, 56
CASE, 41
CASS (mot), 210
Cassette, lecture en mode éditeur, 194
Cassettes et disquettes, 159-178
Catalogue (de disquette), 206
Cellules, dans la pile, 9
CF (code field ou champ code), 124, 218, 222, 244
Champ lien, 221, 245
Champ nom, 220-221, 245
Champ paramètre, 222, 245
CHARTAB, 82
Chaînages, dictionnaire, 226
Chaîne de caractères, 138, 152, 244
Chaîne vide, 75
Chaîne
  de caractères, 73-92
  définition, 89
  recherche de, 87
  temporaires, 74
CHR$, 83
Clavier, 137-138, 149, 151, 259-263
  buffer des entrées, 238
CLS, 97
CMOVE, 126
Code
  champ, 222
  FORTH, 246
  field (cf), 218
Code machine, 245
COLD, 183
COLOR, 97
Coloration du son (pitch), 114-115
COLOROFF, 97
COLORON, 98
Communications, 142, 153, 245, 246, 251
Comparaisons numériques, 20
Compilation et interprétation, 49-72
Compilation, 246
  adresse, 243
  sortie de, 70
COMPILE, 57
Concaténation (chaîne), 77
COND, notation, 10
Condition, 24-26, 246
CONSOLE, 145
CONSTANT, 57
Constantes, 10
Context (variable), 225-226
Context, 57, 126
Contrôleur de communication, 246
Conventions de notation (pile), 10
Conversion de chaîne, 89
Conversion et formatage des entrées-sorties numériques, 74
Conversions entre données numériques et alphanumériques, 73-92
Convert, 83
Copie d'écran, 154
COPY-BLOCKS, 210
Couleur (avec le crayon optique), 149
Couleur, 89-90
Count, 83
CR, 146
Crayon optique, 104, 137-138, 149-150, 153
CREATE, 58
CSP, 58
CSRLIN, 98
CSROFF, 98
CSRON, 99
CTABLE, 58
CURRENT (variable), 225-226
CURRENT, 59
Cyan, 99

D

D (double longueur), notation, 10
D!, 126
D*, 27
D+, 27

D+-, 27
D-, 27
D-err, 168
D., 146
D.R, 146
D0=, 27
D<, 28
D=, 28
D, 126
DABS, 28
DARK, 99
DCONSTANT, 59
DDROP, (suppression d'un nombre), 12
DDUP, (duplication d'un nombre), 12
Debugging, 181
Décimal, 83, 146
Défilement de l'écran, 106
Définitions, 60
Définition FORTH, 49-50
Définition de chaîne, 89
Définition de nouveau mots, 49-72
Définition, (: et ;), 52-53
Defgr, 84
Density, 210
Densité des disquettes, 211
Depth, 12
Dictionnaire, 246
  chaînages, 226
  organisation en mémorie, 220
  organisation en vocabulaires, 224-227
DIGIT, 85
DIR, 212
Directory, 204-207
  affichage, 212
DISK (mot), 211
Disquette, 211
  densité, 211
  formatage, 213-214
  lecture en mode éditeur, 194
  occupation, 212
  organisation physique, 205-207
DLITERAL, 60
DMA, 168
DMAX, 28
DMIN, 28
DN*, 29
DNAM, 168
DNEGATE, 29
DO+, 116
DO, 41, 115
DOES, 60
Données, pile, 9
DOS, 169
DOS, mots, 238
DOVER, 12
DP, 126
DPL, 147
Drive (mot), 212
DROP, 13
DROT, 13
DSWAP, 13
DU<, 29
Dump, 71, 127
DUP, duplication d'une valeur, 11, 13
DUR, 111, 116
DVARIABLE, 62

E

Echange octets poids fort/faible, 11
Ecran, 89, 145, 247
  copie de (hard-copy), 154
Ecrans de l'éditeur, 159
Ecriture de secteur, 215
Ecriture sur disquette, 207
Edit, 183
Editeur, 193-201
  mots de, 199-201
  écrans, 159
EDITOR, 62, 128
ELSE, 43
EMIT, 147
EMPTY-BUFFERS, 169
ENCLOSE, 85
ENDCASE, 43
ENDOF, 43
Entrée du terminal, buffer, 244
Entrées-sorties numériques, 74
Entrées-sorties, 137-157, 172-173, 247
Entrées-sorties, buffer, 239
EOF, 169
ERASE, 128
ERR, 184
ERROR, 183
EXECUTE, 62
EXIT, 43
EXPECT, 148
Extensions de fichier, 206

F

FASC!, 170
FAT, 212
FENCE, 129
Fichier disquette, format, 212
Fichiers, 170-171
  création, 161
  nom par défaut, 238
  sur disquette, 206
  système, 206
File allocation table, 206, 212
FILL, 129
FIND, 130
FIRST, 130, 170
Flot d'entrée, 247
FNAM!, 170
FNB!, 171
Fonctionnement de l'éditeur, 194-199
FORGET, 130
FORMAT (mot), 213-214
Formatage (disquette), 213-214
Forth, 62, 131
  code, 246
  page zéro en mémoire, 238
Fourier, transformation de, 19
Frame, 99

G

Gammes, 109
GCOLOR, 99
Gestion de vocabulaires, 225
Gestion des messages d'erreur, 179-190
GETC, 171
Global, 100
Glossaire général des mots de la cartouche ROM, 9-189
GO, 185
GODO, 36, 44
Graphique et couleur, 93-107
GREEN, 100
Hard copy, 154
HERE, 131
HEX, 86, 148
HILO, 14
HLD, 86
HOLD, 76, 86
HOME, 100

I

I, 45
ID., 131
IF, 45
Immediat, 63, 247
Imprimante, 141, 153
IN?, 148
Indice de boucle, 45
Initialisation de disquette, 213-214
INITMUS, 111
INITSCR, 100
INK, 101
INKEY, 149
INPEN, 149
INPUT$, 150
INPUT, 150
INPUTPEN, 151
INSTR, 87
Interface, 248
INTERPRET, 63
Interpréteur externe, 223-224, 248
Interpréteur interne, 223-224, 248
Interpréteur, 51, 217-220
Introduction, 5
INTRP, 51
INVCOLOR, 101
IO, 172
IOERR, 173

J K

J, 45
KEY, 151
KILL (mot), 214

L

L (longueur d'une chaîne), notation, 10
LSCR, 185
La pile des données, 9-18
Langage machine, 245
LATEST, 132
LEAVE, 45
Lecteur de disquettes, 160, 203, 211, 212
Lecteur-enregistreur de cassettes, 160, 163, 210
Lecture de block sur cassette, 194
Lecture de secteur, 214
Lecture sur disquette, 208
Lecture/écriture de block, 175
LEFT$, 87
LEN, 87
LF (Link field ou champ lien), 218, 245
LFA, 132
Lien, 243
Lien, champ, 221, 245
LIMIT, 132
Line, 101
LINETO, 101
Link field (lf), 218
LINPUT$, 152
LINPUT, 152
LIST, 186
LIT, 64
Litéral, 64-65, 248
LOAD, 173
Local, 101
Location, 101
LOOP, 36-37, 46

M

M*, 29
M/, 29
M/MOD, 30
MAGENTA, 103
Manettes de jeu, 137, 155
MASK, 104
Masquage de bits, 21
MAX, 30
MAXBUF, 174
Message, 186
Messages d'erreur, 184, 264-266
MID$, 87
MIN, 30
Mise en service du système, 241
Mise à zéro des buffers, 169
Mise à zéro, mémoire, 128
MLEN, 88
MOD, 30
Mode
  compilation, 49-50
  exécution, 49-50
  masqué, 103
Moniteur, page zéro en mémoire, 237
Mot
  description, 217-220
  définition, 246, 249
  longueur, 135
  recherche, 217
  transportabilité, 251

MOTOROFF, 174
MOTORON, 174
Mots de la disquette système, 203-215
Mots du FORTH, 253-258
Mots standards de FORTH, 36
Mots système, 36, 229-232
MOVE, 133
MUSBUF, 112
Musique, 109-117
MUSTAB, 112
Mémoire de masse, 159-178
Mémoire utilisateur, 235-236
Mémoire
  affichage (dump), 127-128
  nombre de buffers, 159
  opérations sur, 119-135
  organisation, 233-240

N

N, notation, 10
Name field (nf), 218, 245
NEGATE, 31
NF (Name field), 218, 245
NFA, 133
Nom, champ, 220-221, 245
NOMASK, 103
Nombre entier, conversion et formatage, 75-76
Nombres "flottants", 19
Nombres 16 et 32 bits, 20, 138, 249
NOT, 31
Notation (effets sur la pile), 10
NOTE, 113
Notes, dans la table musicale, 116-118
Number, 88

O

Occupation de la mémoire (tableau), 234
OCT, 114
OF, 47
OFFSET, 175
ONERR, 186
ONPEN, 153
Opérations logiques, 20-21
Opérations sur la mémoire, 119-135
OR, 31
Organisation de la mémoire, 233-240
OVER, 14

P

PAD, 88
Page, 104
PALE, 104
PAPER, 104
Parameter field, 219
Parameter stack (voir pile utilisateur ou pile des données), 9
Parameter stack, 250
Paramètre, champ, 222, 245
Paramètres de la pile, description, 73
Pause, 117
PEN, 104
Périphériques de sortie, 186
Périphériques, 250
Périphériques, stockage, 159-178
Périphériques, zones mémoires, 233
Personnalisation de l'éditeur, 201
PF (Parameter field), 219, 245
PFA, 133
PICK, 14
Pile "système", 9
Pile de retours, 9
Pile des données, 9-18, 223-224, 238-239, 250
Pile des retours, 11, 223-224, 239, 250
Pile utilisateur, 9
Pile, effet sur, 10
Pistes et secteurs, 205-206
PITCH, 114, 117
PLAY, 115
POINT, 104
Pointeur de programme, 223
Port de communication, 142, 153, 160, 176, 250
POS, 105
PREV, 175
PRINTER, 153
Protection, mémoire, 129
PSET, 106
PTRIG, 154
PUTC, 175

Q R

QUERY, 164
QUIT, 187
R/W, 175
R0, 14
R>, 14
R, 15
RS-232, 250
RAM système, 237-238
RAM utilisateur, 236
Recherche d'une chaîne, 87
Recherche, dictionnaire, 130
RED, 106
REDIT, 187
Redémarrage à froid, 183
Registres, 187
RENUM, 214
Renumérotation de blocks, 214
REPEAT, 47
Retours, pile de, 9
Return stack, 250
RIGHT$, 89
RND, 187
ROLL, 15
ROT, 15
RP!, 16
RP0, 16
RP, 16
RS-B, 176
RS-S, 176
RUN, 188
Réinitialisation des buffers, 167

S

S->D, 32
S0, 17
Saut de fichier, 177
Sauvegarde (cassette ou disquette), 188
SAVE, 188
SAVE-BUFFERS, 176
SCR, 189
SCREEN, 106
SCROLL, 106
SCRPRT, 154
SEC-RD, 214
SEC-WR, 215
Secteur, lecture, 214
Secteur, écriture, 215
SIGN, 76, 89
SIZE, 107
SKIP, 177
SKIPB, 177
SMUDGE, 65-66
Son, coloration, 114-115
Sortie de boucle, 45-46
Sortie de mode compilation, 70
Source, 251
SP!, 17
SP0, 17
SP, 17
Space(s), 154-155
SQUISH, 18
STATE, 66
STICK (manette de jeu), 155
Stockage de masse, 159-178, 251
STR (chaîne de caractères), notation, 10
STR$, 89
STRING, 66, 89
Structure de contrôle, 35-48
Structure du dictionnaire, 217-227
Structures conditionnelles, 35
Structures répétitives, 35
SUB$, 90
Suppression d'un nombre (pile), 12
Suppression de block, 214
SWAP, 18
SWAPDROP, 18
Système d'exploitation, 160
Système, informations sur disquette, 206
Système, pile, 9
Sysvec, 189, 231
Sélecteur numérique (CASE), 41
Série, port, 153

T

Table, 67
  d'allocation des secteurs, disquette, 239
  des vocabulaires, 225
Table musicale, 109
  structure, 113
Table, caractères graphiques, 82
Tableaux, (arrays), 21, 56
TEMP, 115, 117
Terminaison de boucle, 46
Terminal input buffer, 155, 251
TEXT, 90
THEN, 47
THRU, 177
TIB, 155, 251
TOGGLE, 134
Touches de l'éditeur, 195-198
Transfert de blocks, 161
Transmission série, 176, 251
Transportabilité (pour un mot), 251
TRAVERSE, 134
TYP!, 171
Type, 156

U

U*, 32
U., 156
U.R, 156
U/MOD, 33
U<, 33
UD (nombre double longueur non signé), notation, 10
UD., 157
UD.r, 157
UDN*, 33
UN, notation, 10
UNITL, 47
UPDATE, 178
USE, 178
USTR$, 90
Utilisateur, pile, 9
Utilisateur, variables, 252
Utilitaires, 179-190

V

VAL, 91
Valeur de vérité (dup), 11
Valeur de vérité, 252
Variable, 67
Variable "user", 252
Variables chaînes, 73
Vectorisation, 229-232
VLIST, 135
VOC-LINK, 68
Vocabulaire, 68, 252, 224-227
Vrai/faux, notation, 10

W X Y Z

WAIT, 189
WHERE, 189
WHILE, 47
WHITE, 107
WIDTH, 135
WINDOW, 107
WORD, 91
XOR, 33
YELLOW, 107
Zone RAM disponible, 234
Zone RAM système, 234
Zone cartouche ROM FORTH, 233
Zone mémoire, 233-234
  entrées-sorties, 233
  moniteur, 233
  mémoire écran, 234
  mémoire, tableau, 239

Imprimé en France par Pollina, 85400 Luçon - N° 7141 — Dépôt légal : Mai 1985